Optimal placement of fused multiply-add (FMA) instructions.

Konstantin Serebryany
konstantin.s.serebryany@intel.com

Fused multiply-add (FMA) on IPF

`FMA (a,b,c) = +a*b+c;`

`FMS (a,b,c) = +a*b-c;`

`FNMA(a,b,c) = -a*b+c;`

Most compilers apply simple peephole rules to generate FMAs,

i.e. patterns of small size are considered:

```
a*b+c; a*b-c; -a*b+c; -(a*b); (-a)*b;
```

Example: **`res=-a*(b*c-d)-e;`**

gcc 4.0 for IPF generates 3 FMAs

```
t1=FMS(b,c,d); t2=FMS(0,0,t1); res=FMS(a,t2,e);
```

While 2 FMAs are enough: **`res=a*(-b*c+d)-e`**

```
t1=FNMA(b,c,d); res=FMS(a,t1,d);
```

Factorization. CSE. Tree height reduction.

Factorization:

`a*b*c+a*d*e+a*f` ➔ `a*(b*c+d*e+f)`

…not always profitable:

`a*b+a*c+a*d` ➔ `a*(b+c+d)`

CSE:

`a*b*c*b*a` ➔ `t=a*b; t*c*t`

…may hurt:

`(a*b+c)*(a*b+d)` ➔ `t=a*b; (t+c)*(t+d)`

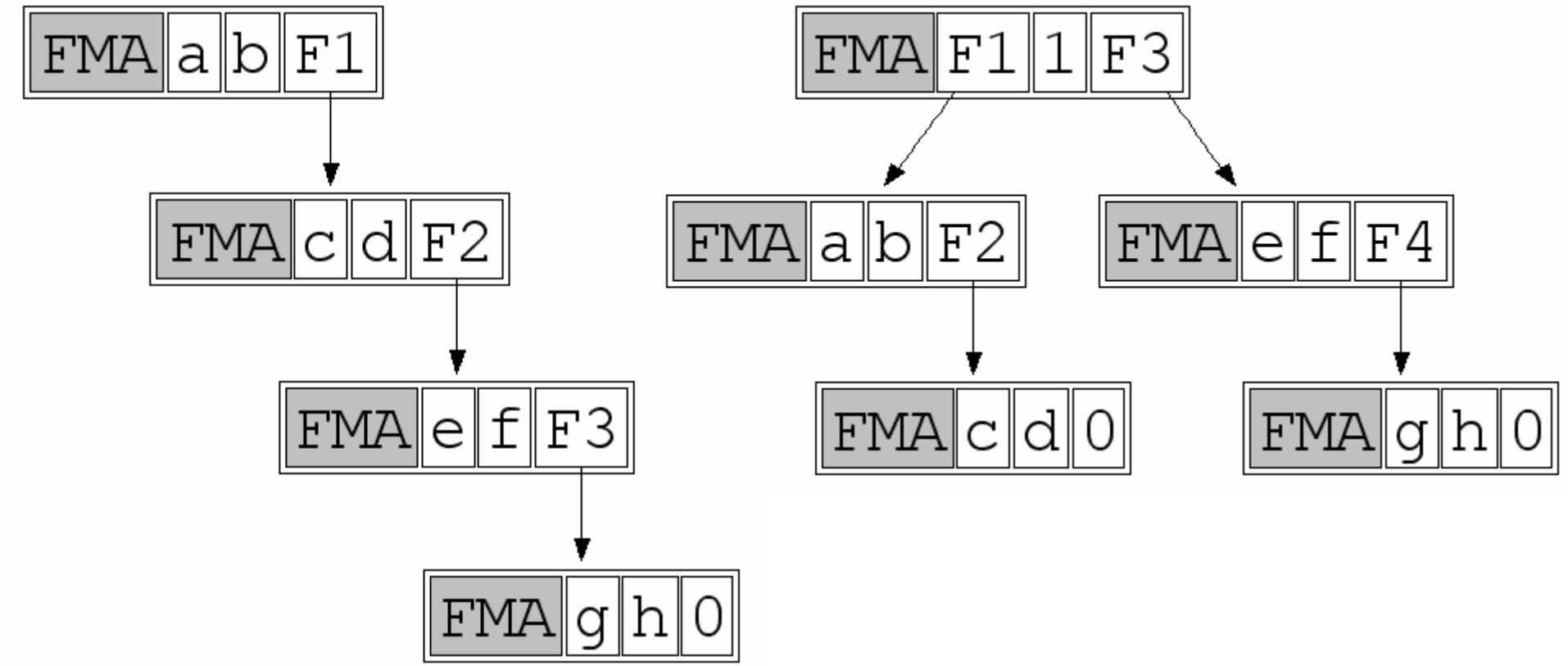
Tree height reduction:

`(a+(b+(c+d)))`➔`(a+b)+(d+e)`

…may increase number of FMAs:

`a*b+(c*d+(e*f+g*h))`➔`(a*b+c*d)+(e*f+g*h)`

THR: latency vs. complexity

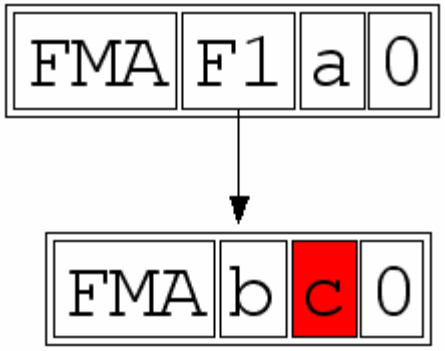


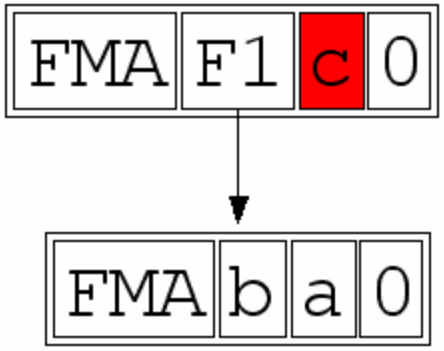
Two ways to generate code for

a*b+c*d+e*f+g*h

Argument availability

```
float abc(float a, float b, float *p){
    float c = *p;
    return a * (b * c);
}
// better:
float abc(float a, float b, float *p){
    float c = *p;
    return c * (a * b);
}
```





Definition of optimality

A sequence of FMAs (an FMA DAG) is the optimal sequence for a given expression, if

1. Complexity (number of FMAs) is minimal.
2. Latency (height of the DAG) is minimal within minimal complexity.
3. Arguments available later are used later (while preserving minimal complexity and latency).

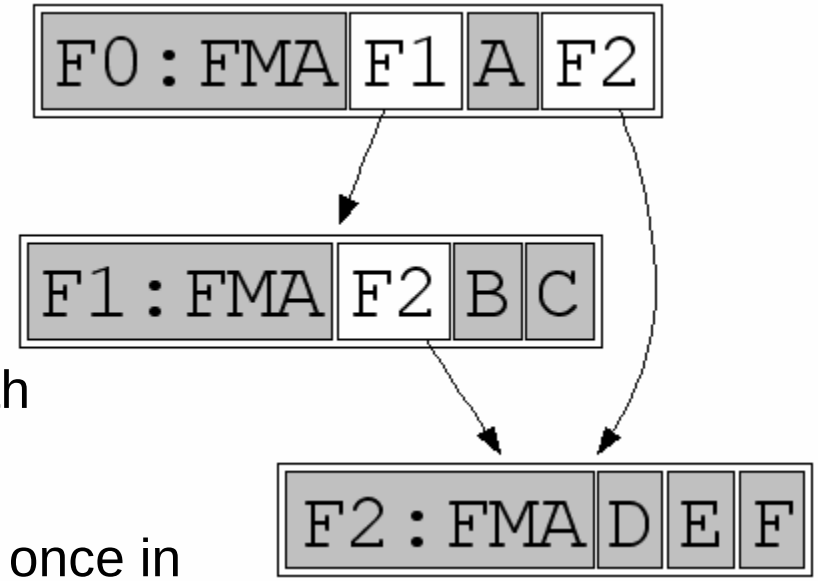
Algorithm: *optimal* placement of FMAs

1. Generate a table of patterns (each pattern is an FMA DAG). Done just once.
2. Match each incoming expression against the generated table of FMA DAGs.

Pattern generation

A sequence of N FMA (not FMS nor FNMA) instructions F0, F1, ... is an **FMA pattern** if

- Instruction Fi may have these operands:
  - instruction Fj , j > I
  - terminal A,B, ...,
  - constant 0 or 1.
- The sequence forms a connected DAG with one root node (F0).



- Each of the terminals A,B, ... appears only once in the sequence and smaller (lexicographically) terminals appear first; if a sequence has M terminals than these are the first M letters of alphabet.

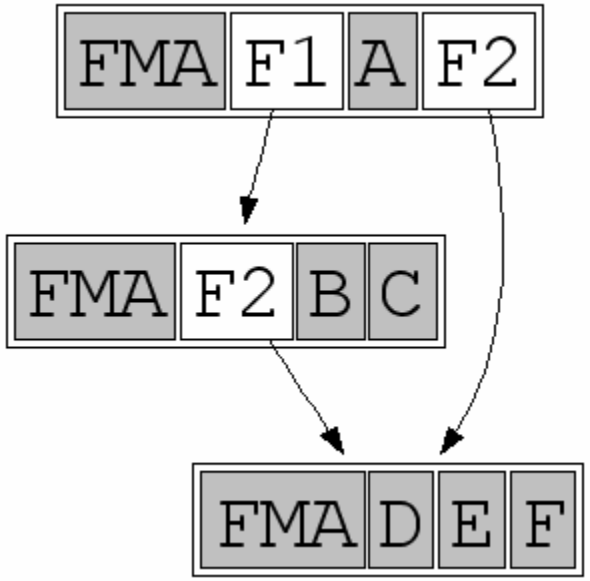
Pattern generation (cont.)

Canonical form of FMA DAG (open all parenthesis):

+ABDE+ABF+AC+DE+F

Shape of FMA DAG (a number in a binary form):

01111011101101101



The pattern generator generates all possible FMA patterns of sizes 5 and less.

It prunes duplicates and redundant DAGs.

All DAGs are sorted according to

- Shape (first)
- Complexity
- Height (last)

Approximately 100000 entries for 5-FMA DAGs.

Pattern matching

Given:

1. Pre-generated table of FMA patterns.
2. An expression consisting of

   operations: **`-,*,+,(…)`**

   and terminals: **`a, b, c`**, ...

   (terminals are ordered according to their availability,

   '**a**' is available later).

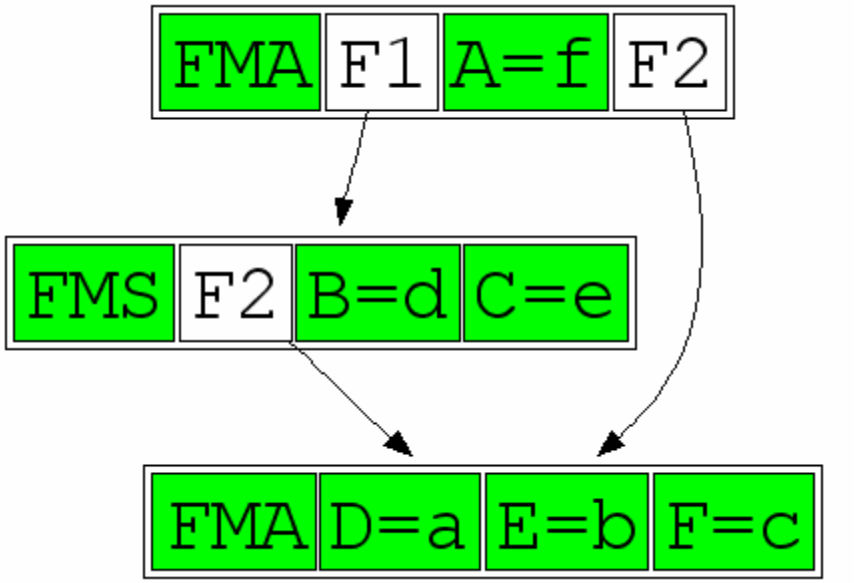
Example: **`+ab+f*(cd-e+abd)+c`**

Find: an equivalent *optimal* sequence of FMA, FMS, FNMA instructions.

Pattern matching (cont.)

1. Compute the canonical form and the shape of the expression:
   +ab+f*(cd-e+abd)+c ➔
   +abdf+cdf+ab-ef+c (canonical form)
   01110111011011 01 (shape)
2. In the generated pattern table, find all patterns with this shape.
3. For each such pattern, try to find a valid mapping between formal and actual terminals and a valid sign combination. The first pattern to match is optimal (since they are sorted by complexity and latency).

Expression:
+abdf+cdf+ab-ef+c
Pattern from the table:
+ABDE+ABF+AC+DE+F
Valid mapping:
A->f; B->d; C->e;
D->a; E->b; F->c;
Valid sign combination:
FMA, FMS, FMA



Pattern matching (cont.)

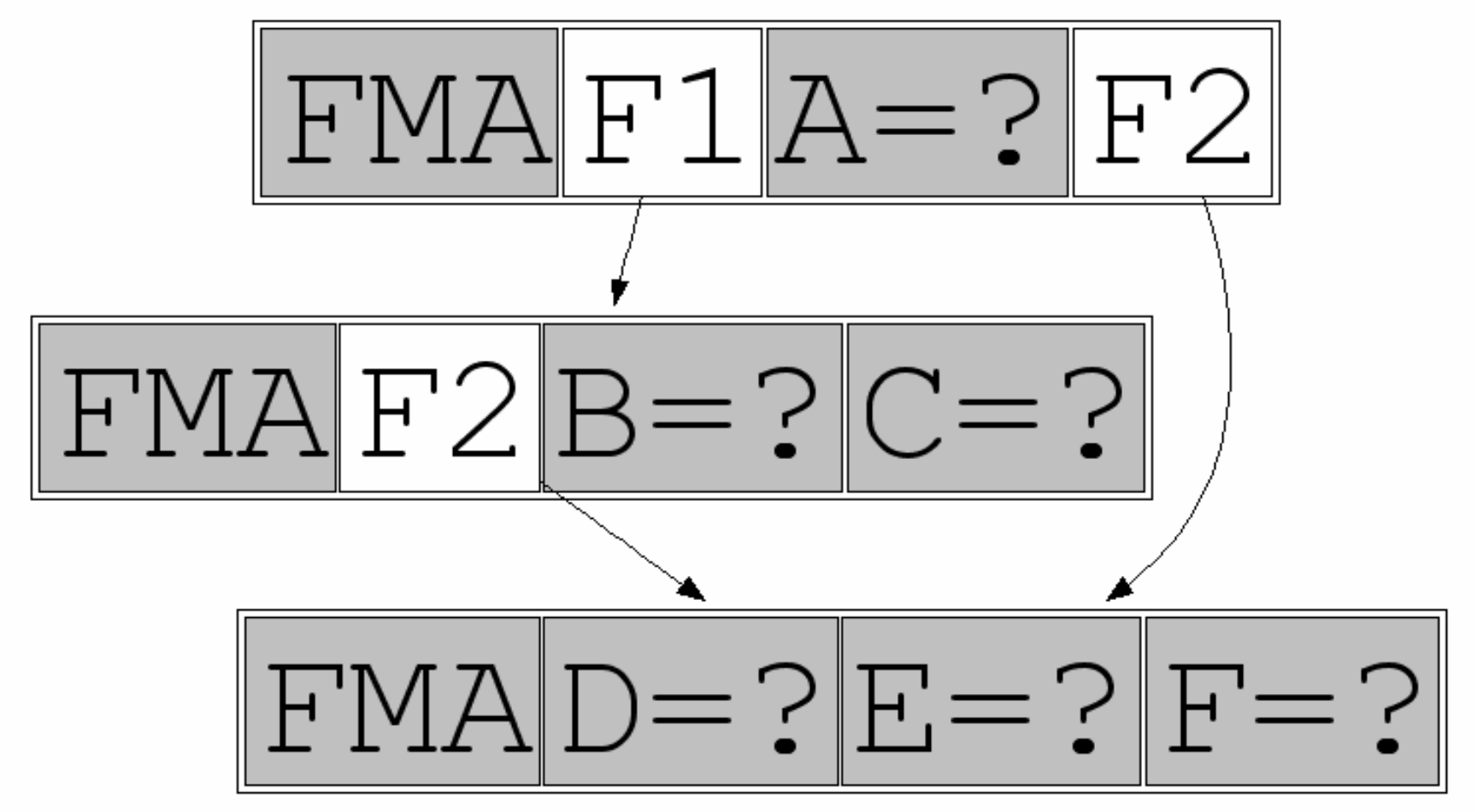
There are N^n mappings between formal and actual terminals.

We used a recursive breadth-first search combined with several constraints which allow to cut the search tree early (similar the ‘8 queens’ problem).

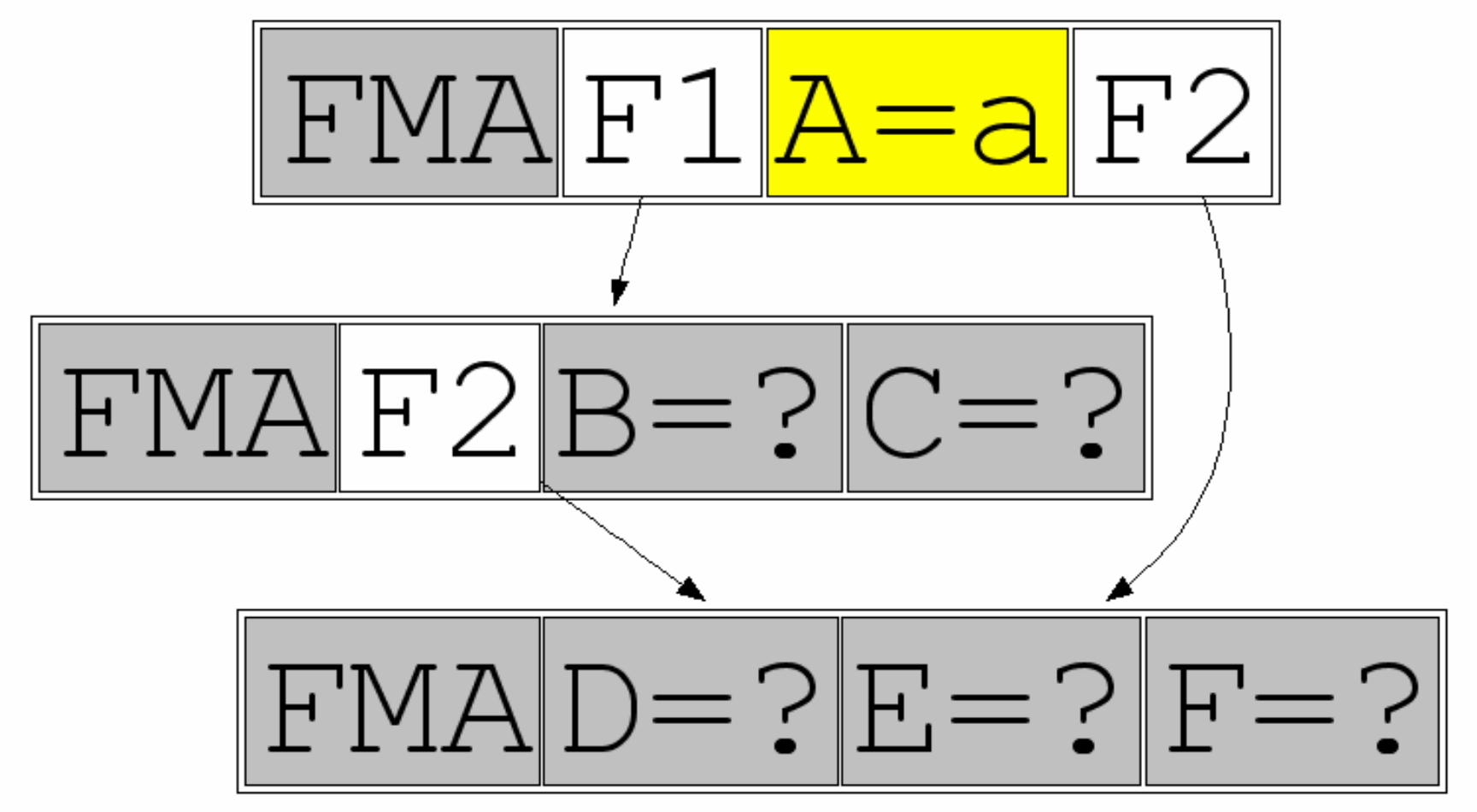
Valid sign combination could be found with exhaustive search (search space is not too large).

Formal: +ABDE+ABF+AC+DE+F

Actual: +abdf+cdf+ab-ef+c

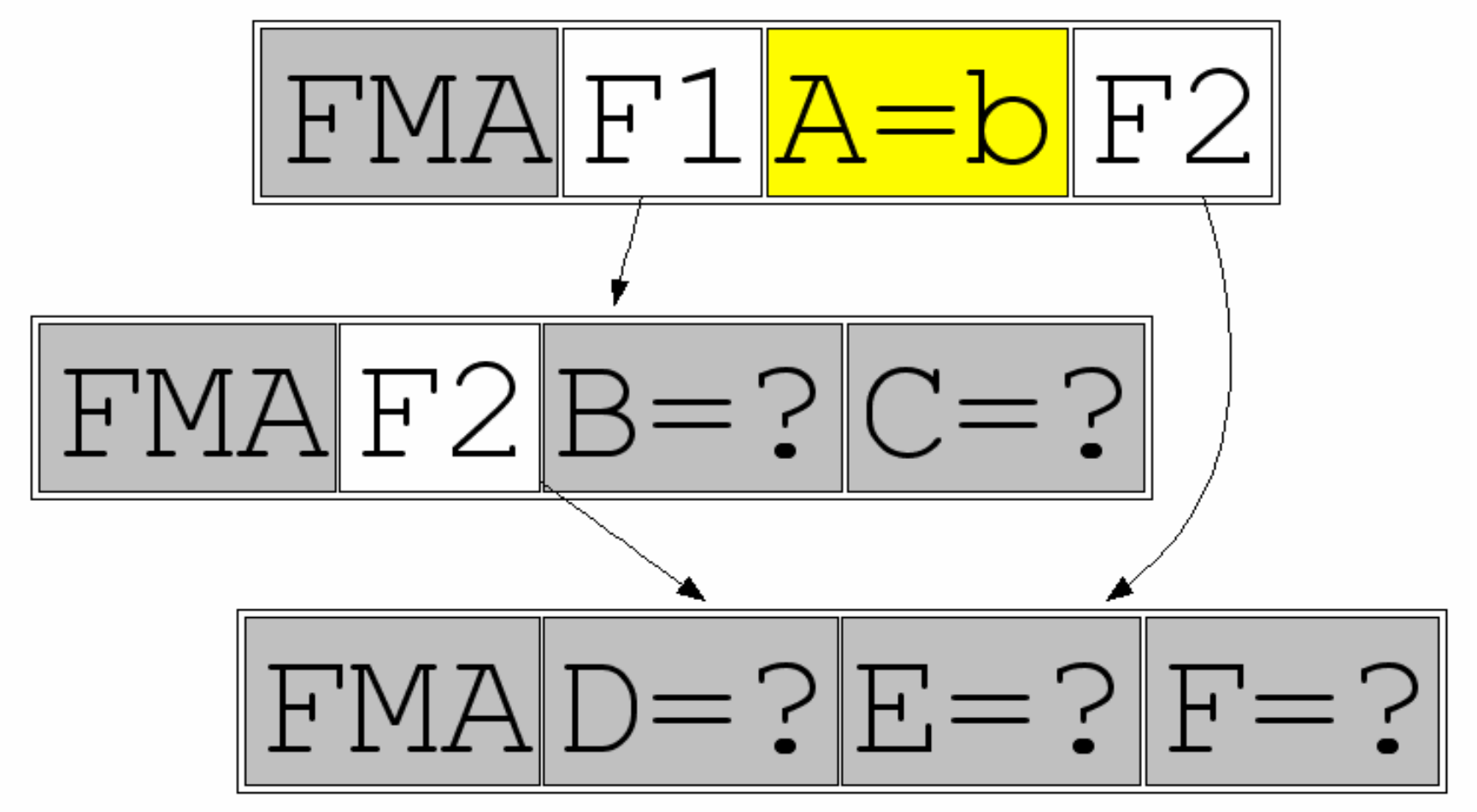


Formal: +ABDE+ABF+AC+DE+F
Actual: +abdf+cdf+ab-ef+c



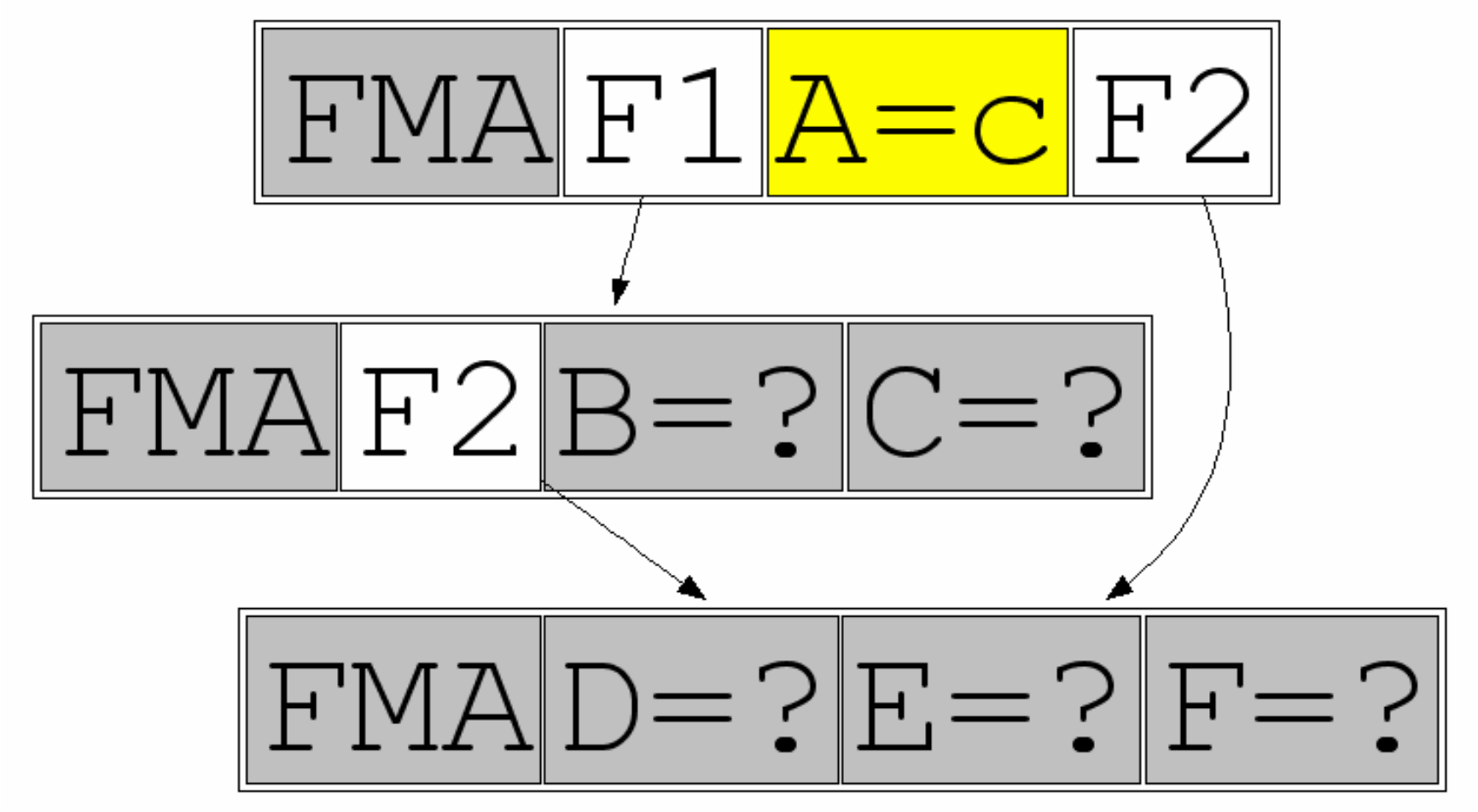
Formal: +ABDE+ABF+AC+DE+F

Actual: +abdf+cdf+ab-ef+c

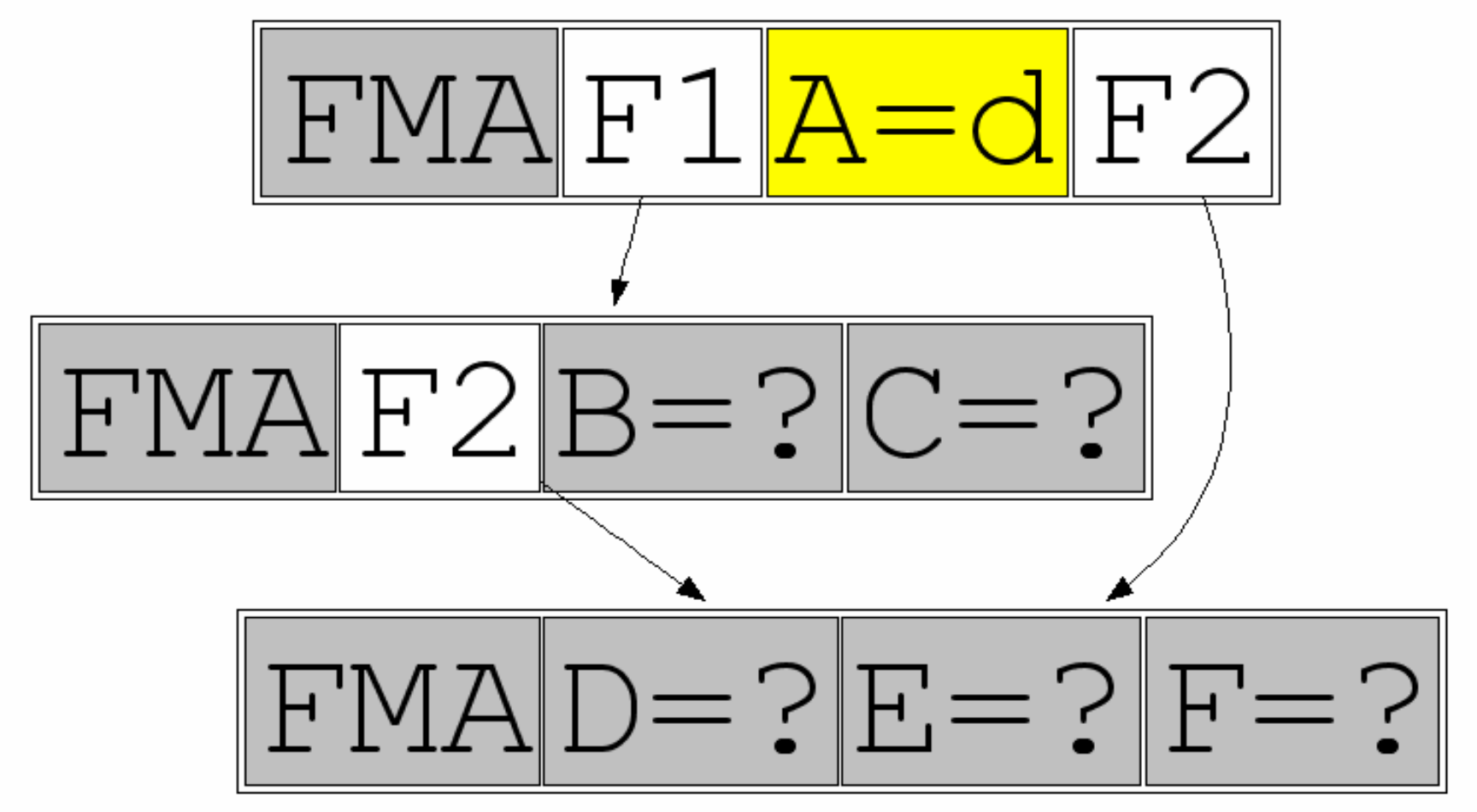


Formal: +ABDE+ABF+AC+DE+F

Actual: +abdf+cdf+ab-ef+c

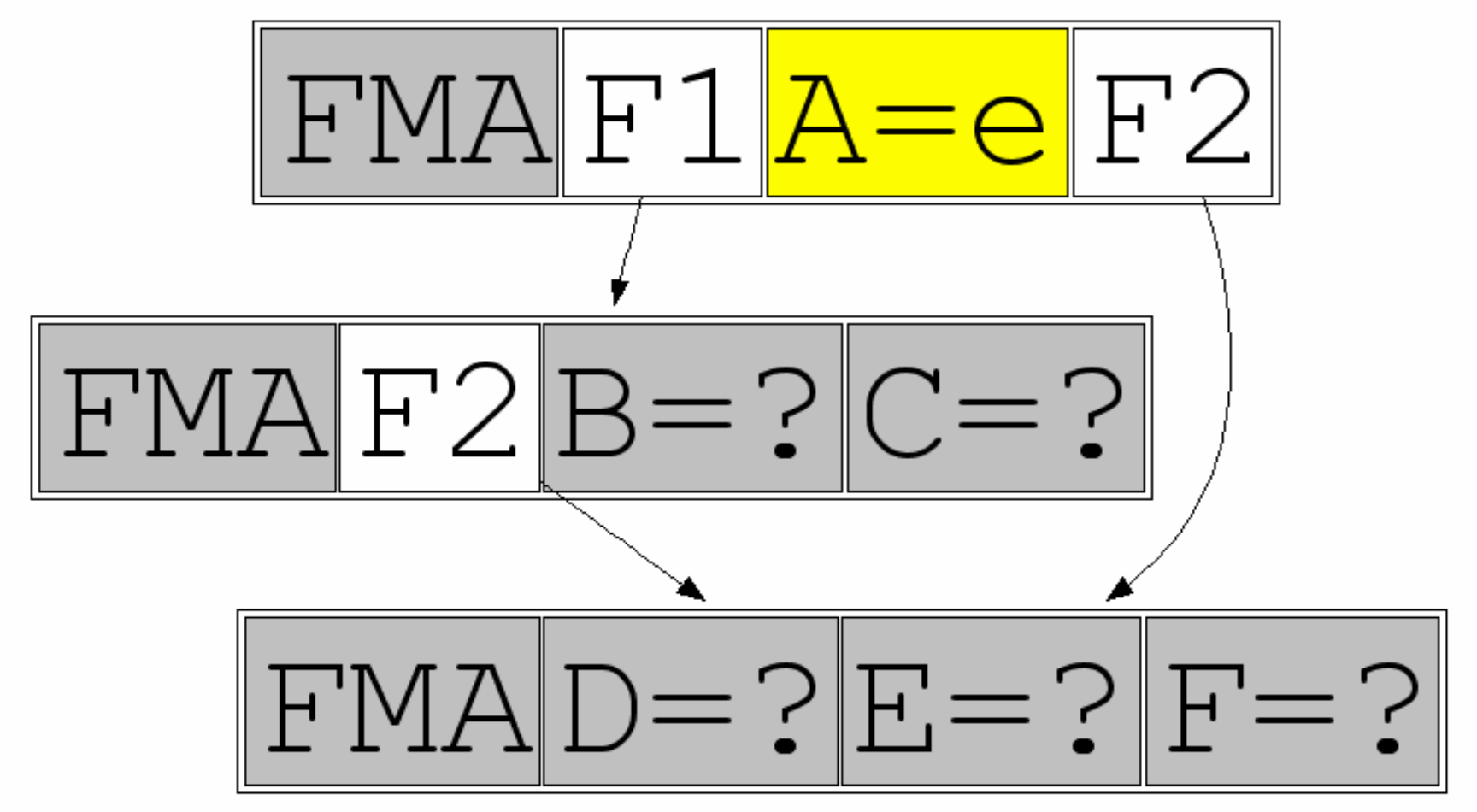


Formal: +ABDE+ABF+AC+DE+F
Actual: +abdf+cdf+ab-ef+c



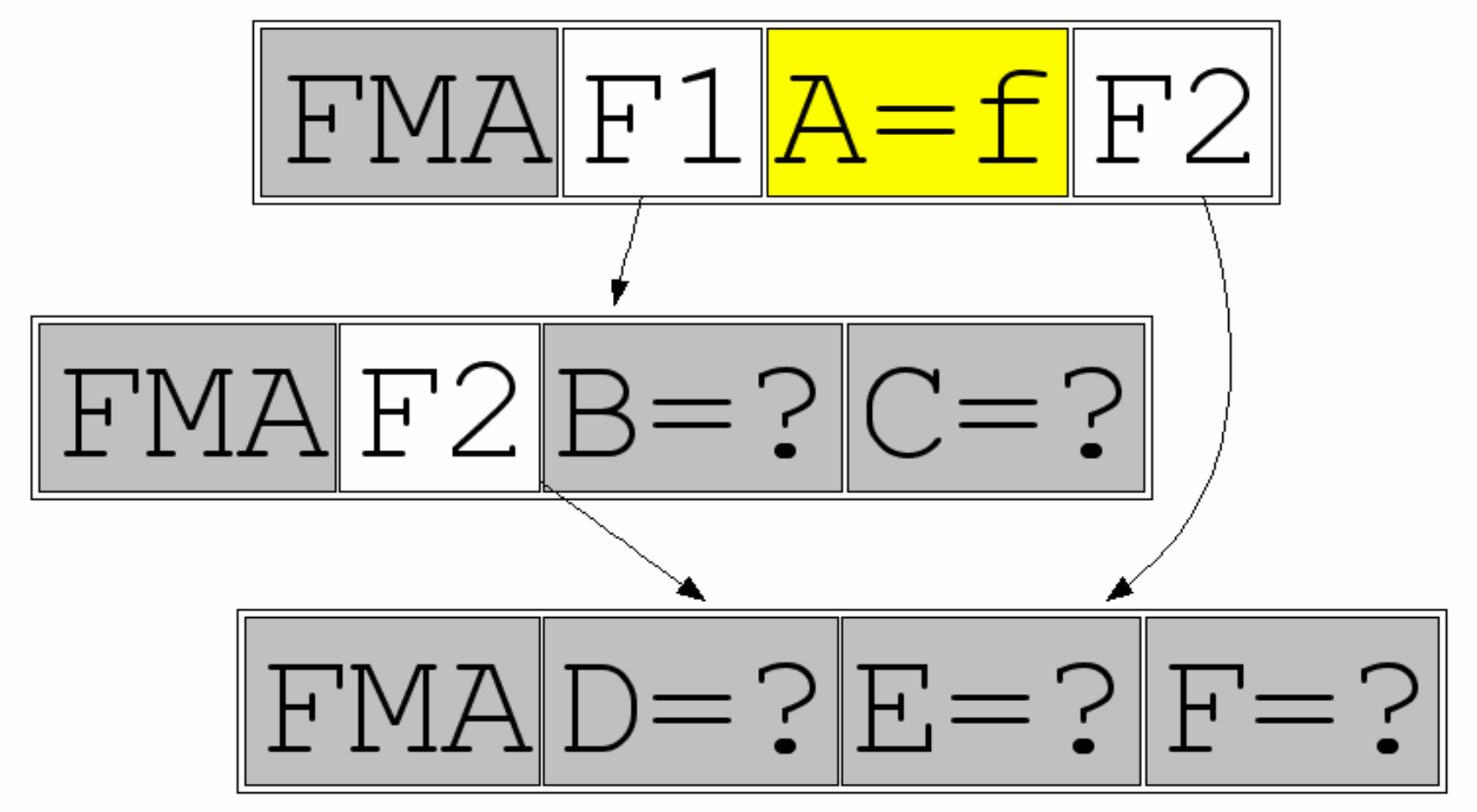
Formal: +ABDE+ABF+AC+DE+F

Actual: +abdf+cdf+ab-ef+c



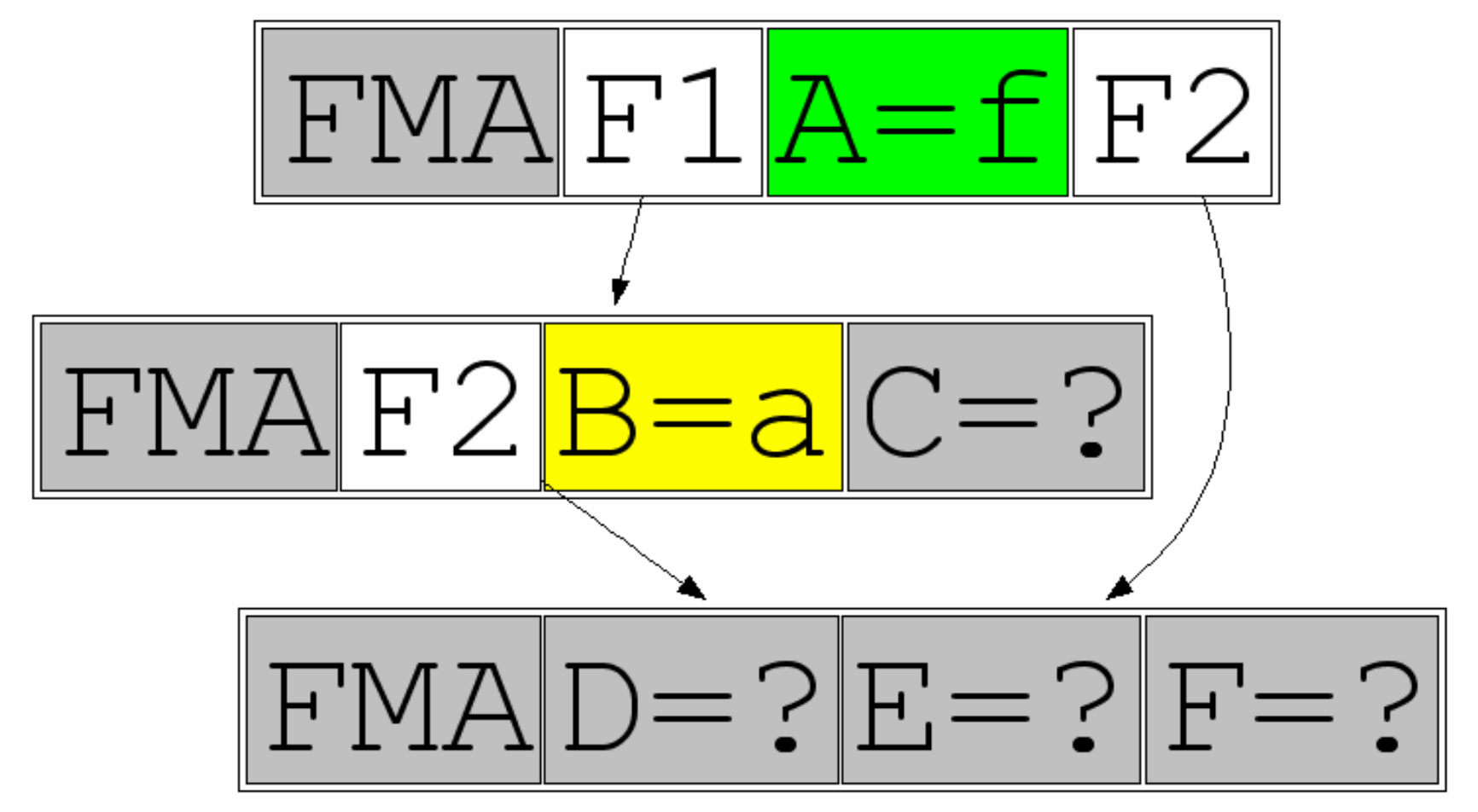
Formal: +ABDE+ABF+AC+DE+F

Actual: +abdf+cdf+ab-ef+c



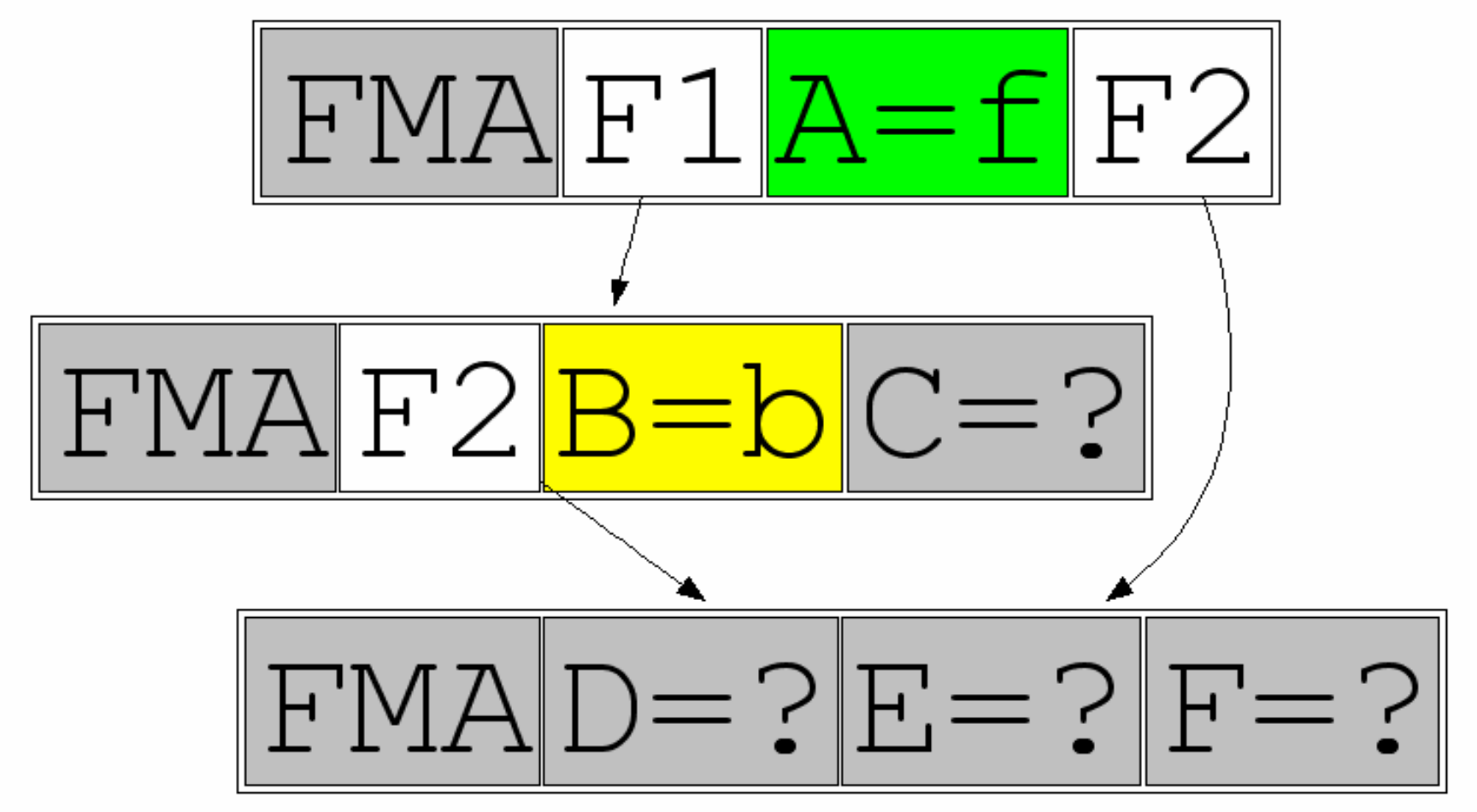
Formal: +ABDE+ABF+AC+DE+F

Actual: +abdf+cdf+ab-ef+c

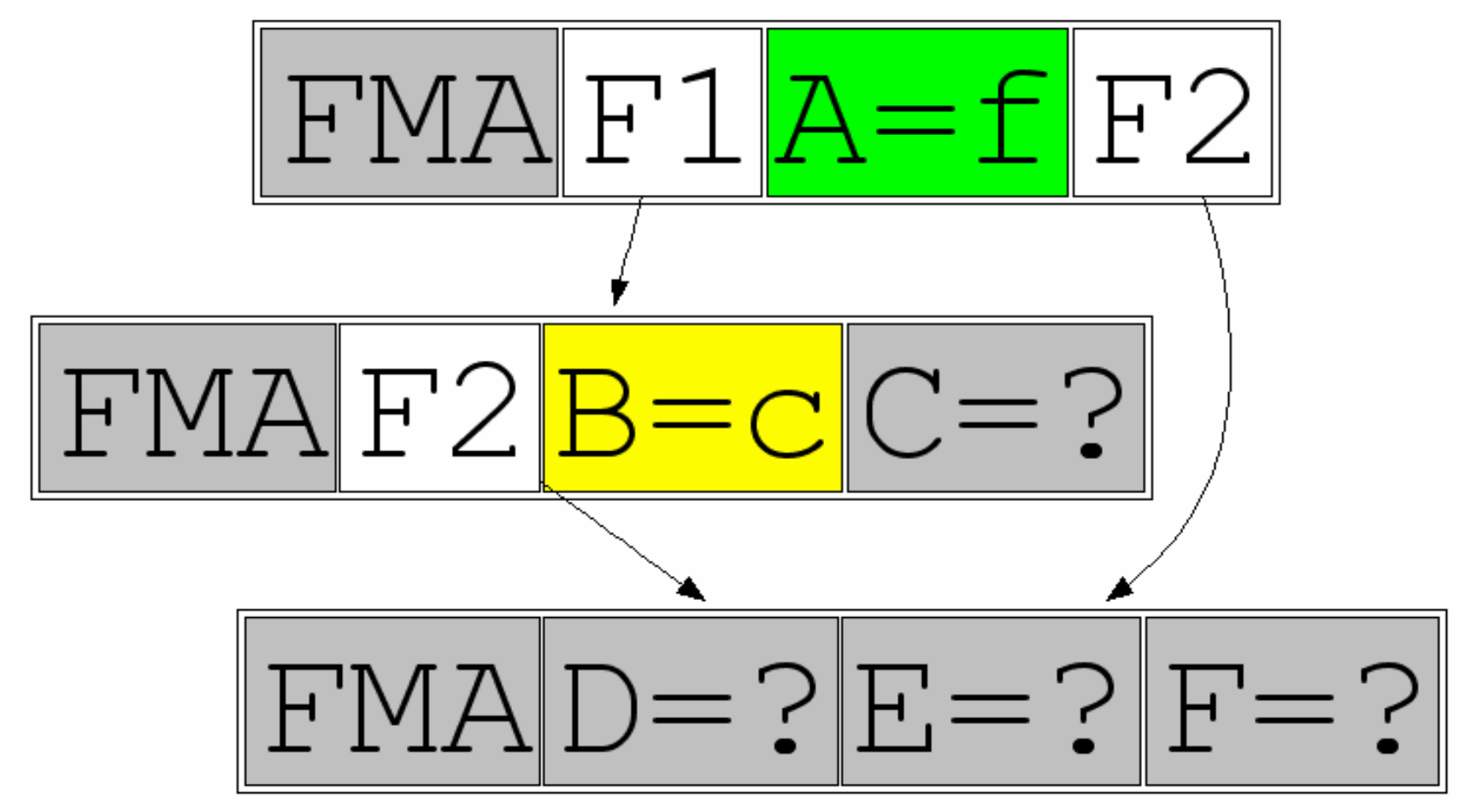


Formal: +ABDE+ABF+AC+DE+F

Actual: +abdf+cdf+ab-ef+c

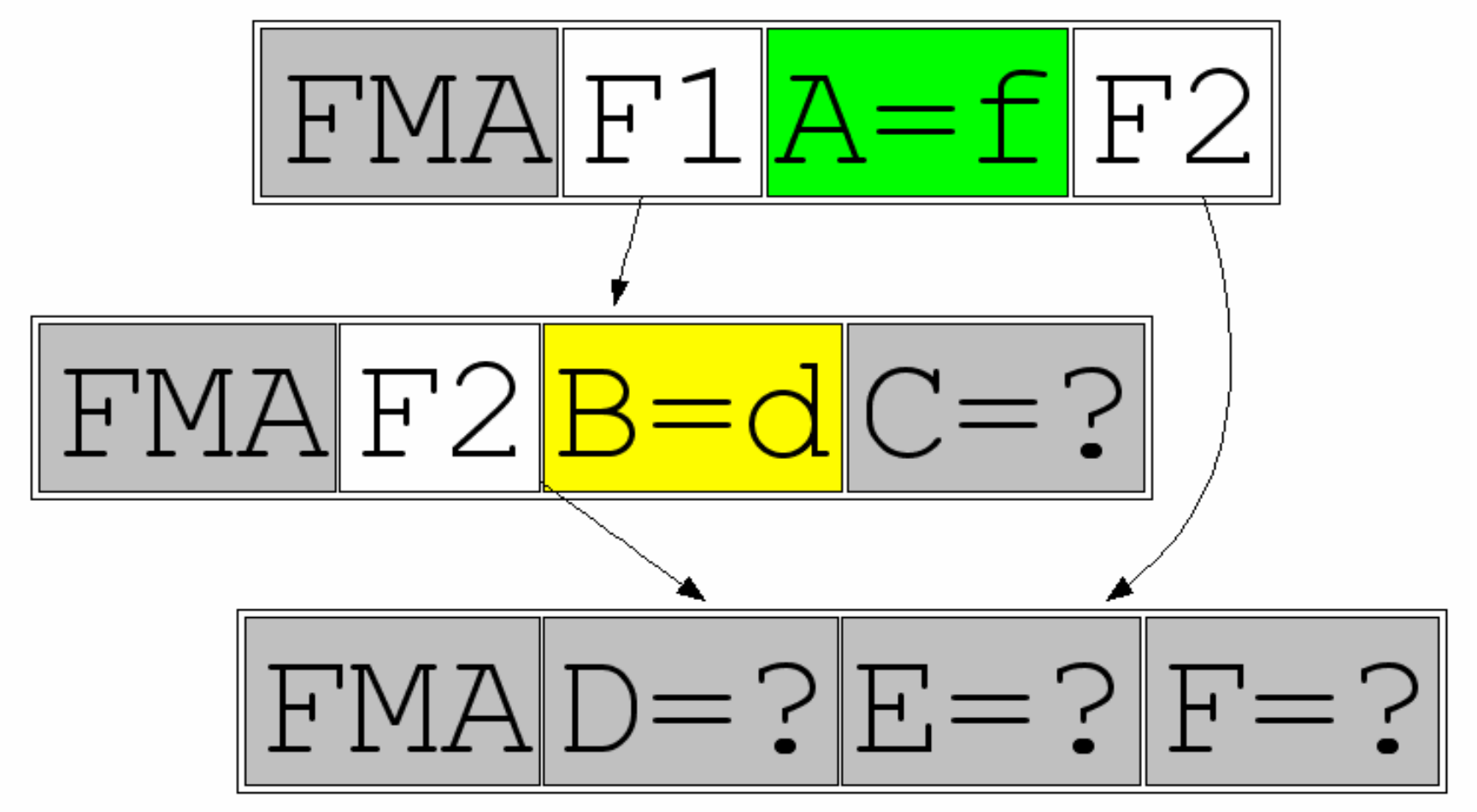


Formal: +ABDE+ABF+AC+DE+F
Actual: +abdf+cdf+ab-ef+c

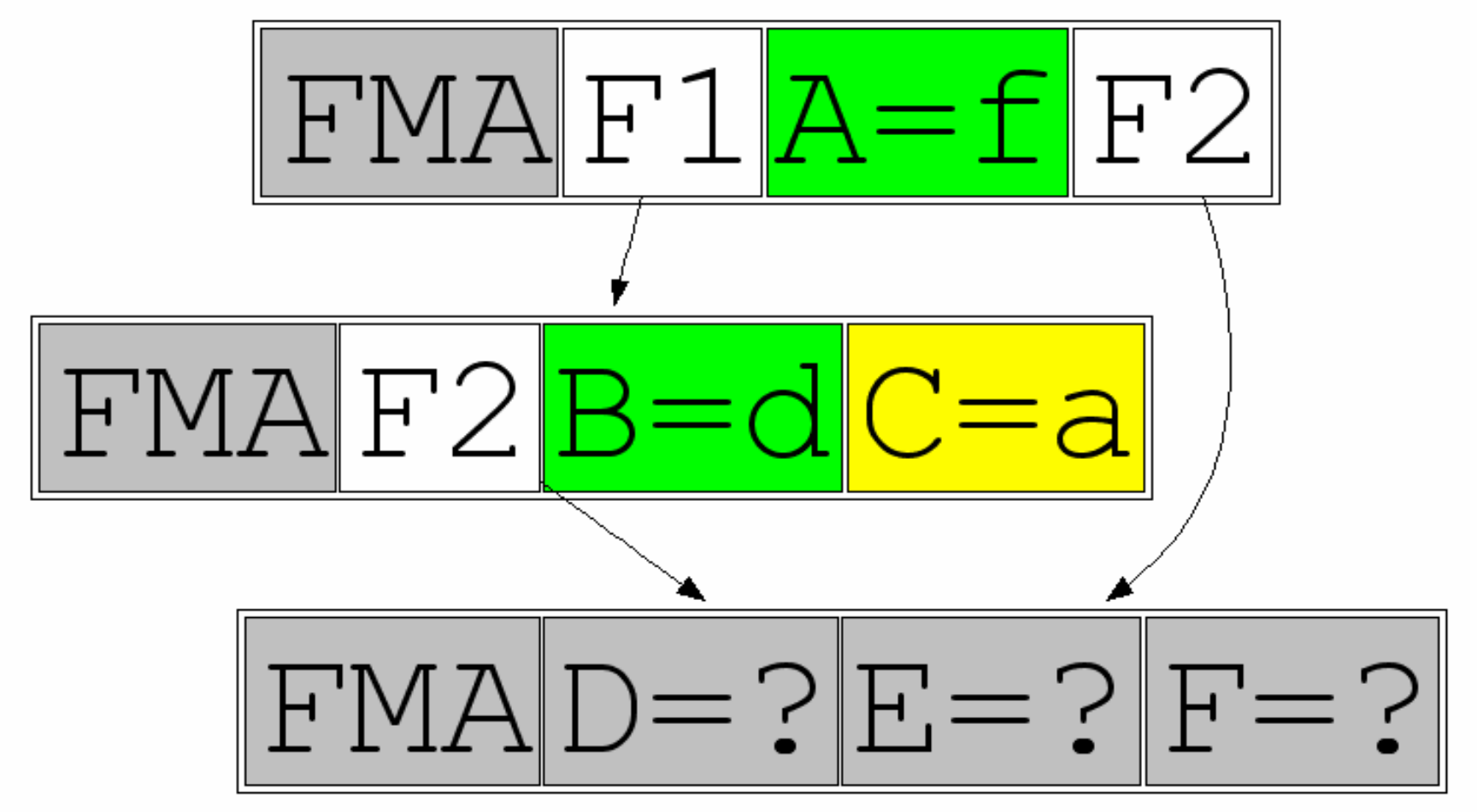


Formal: +ABDE+ABF+AC+DE+F

Actual: +abdf+cdf+ab-ef+c

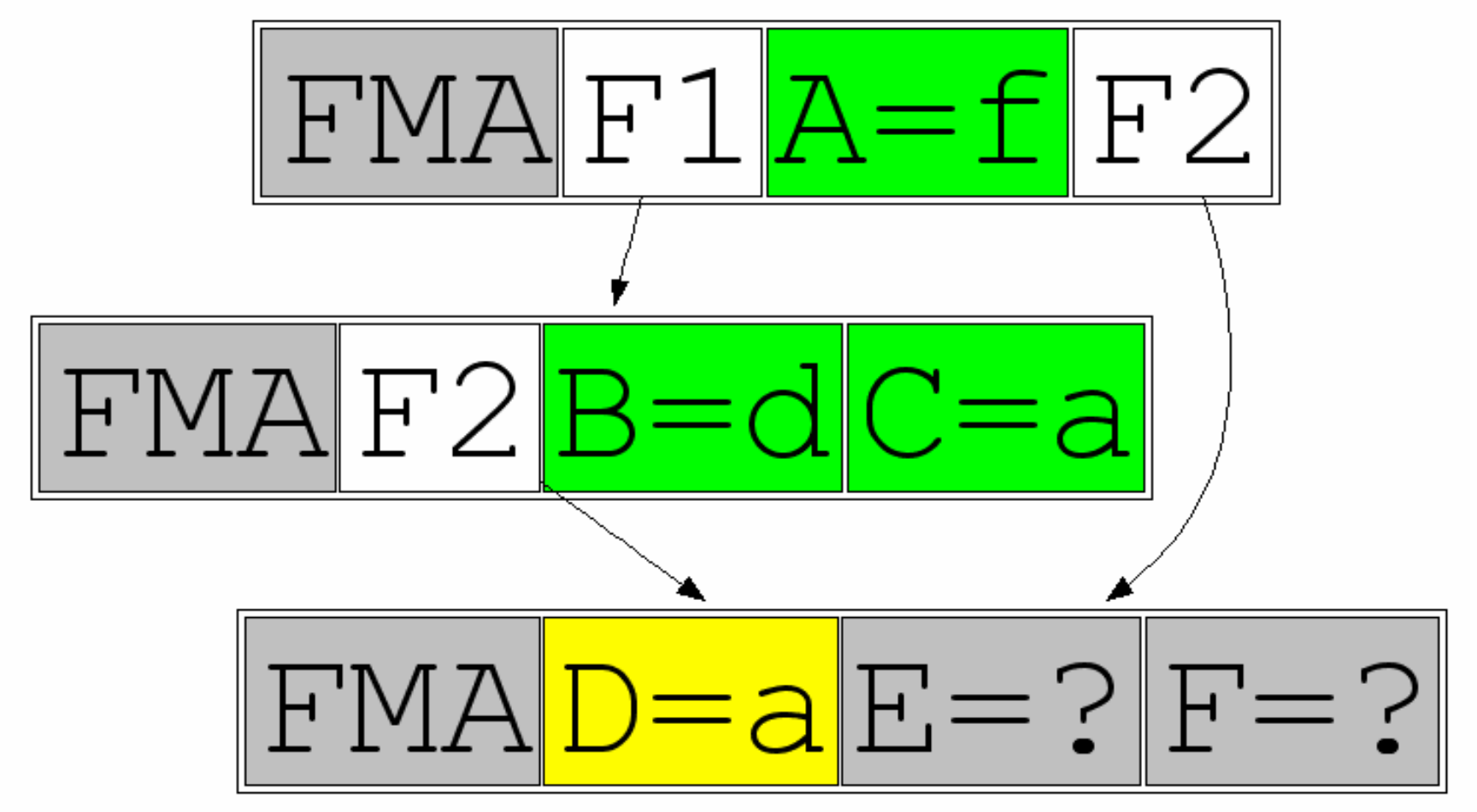


Formal: +ABDE+ABF+AC+DE+F
Actual: +abdf+cdf+ab-ef+c

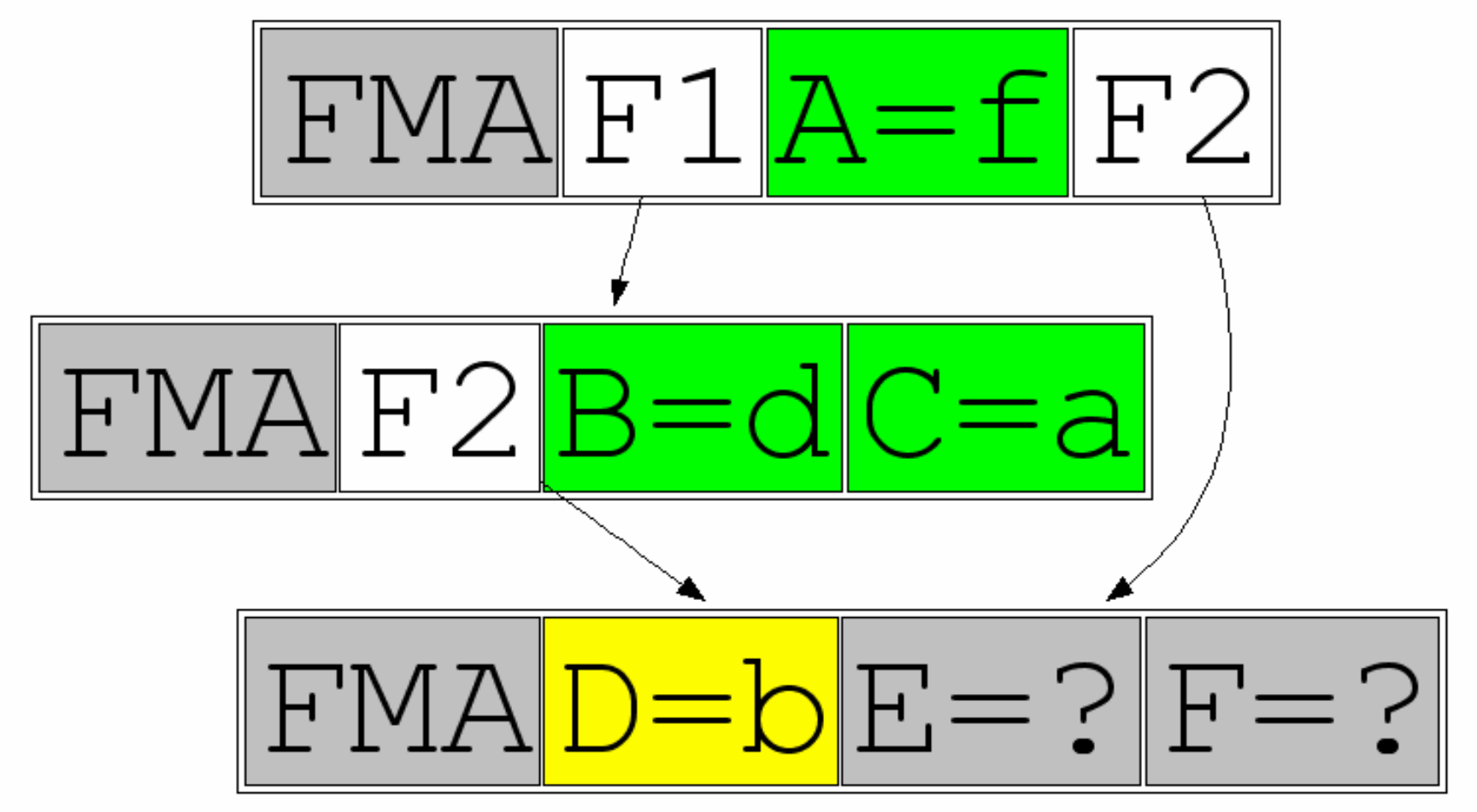


Formal: +ABDE+ABF+AC+DE+F

Actual: +abdf+cdf+ab-ef+c

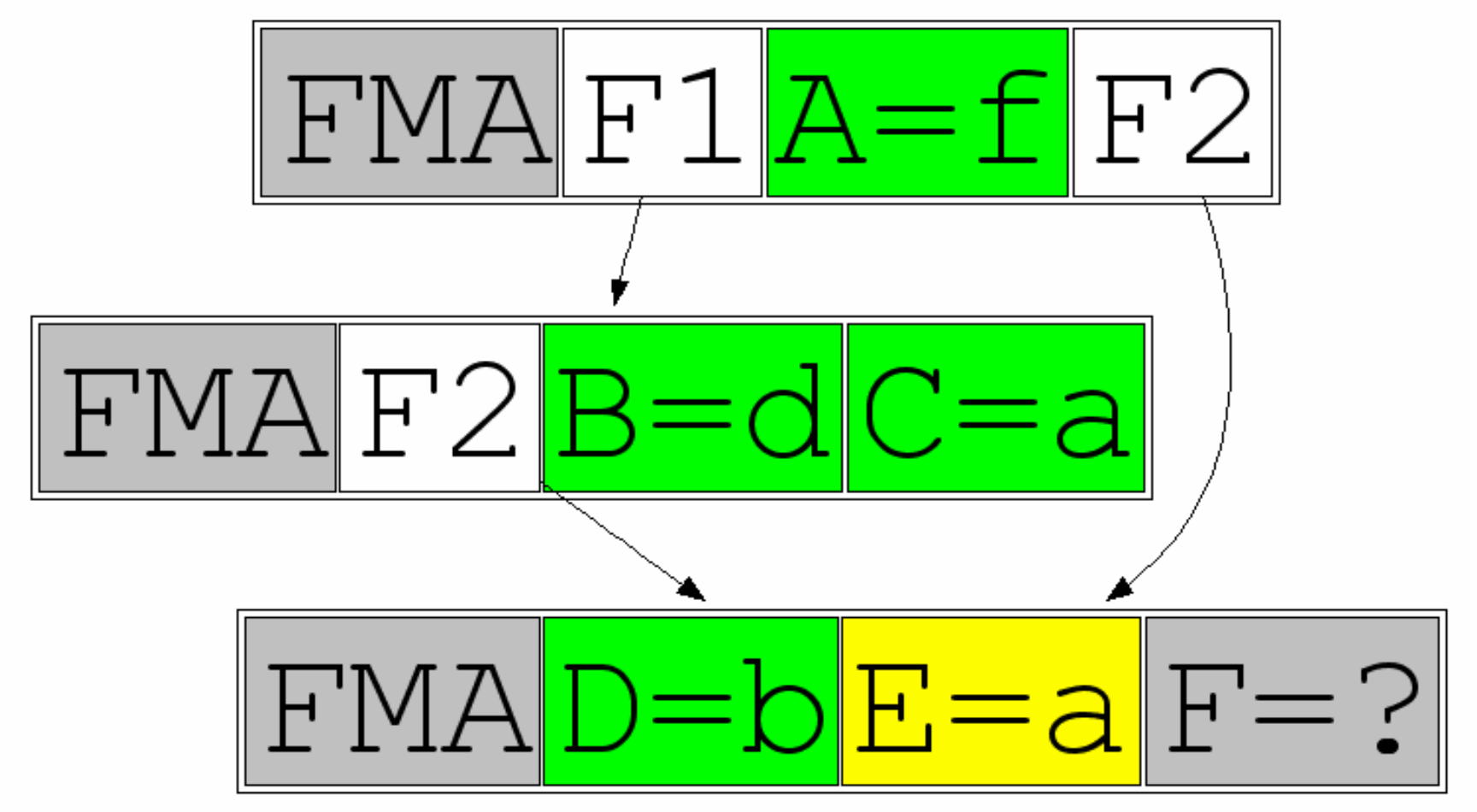


Formal: +ABDE+ABF+AC+DE+F
Actual: +abdf+cdf+ab-ef+c



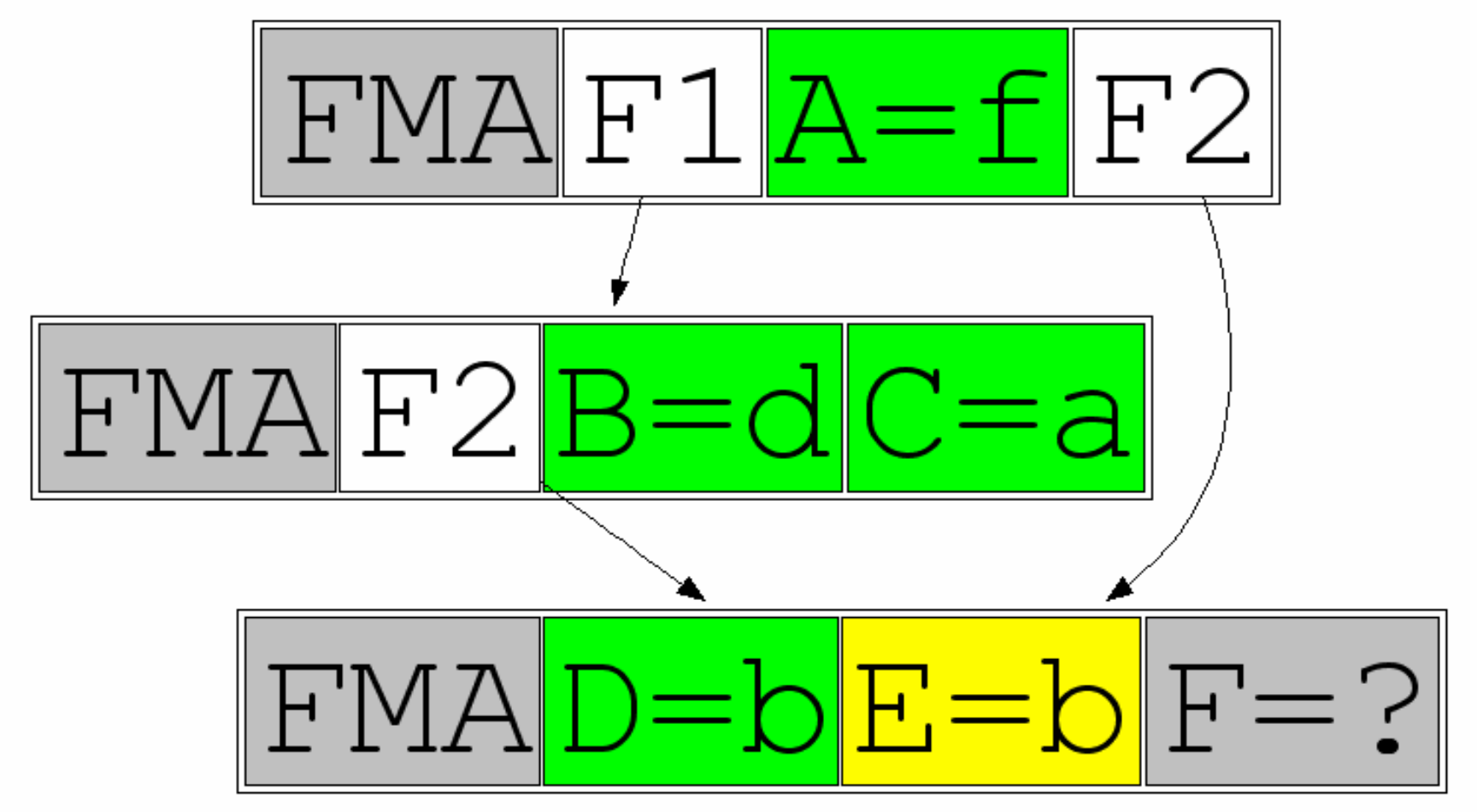
Formal: +ABDE+ABF+AC+DE+F

Actual: +abdf+cdf+ab-ef+c



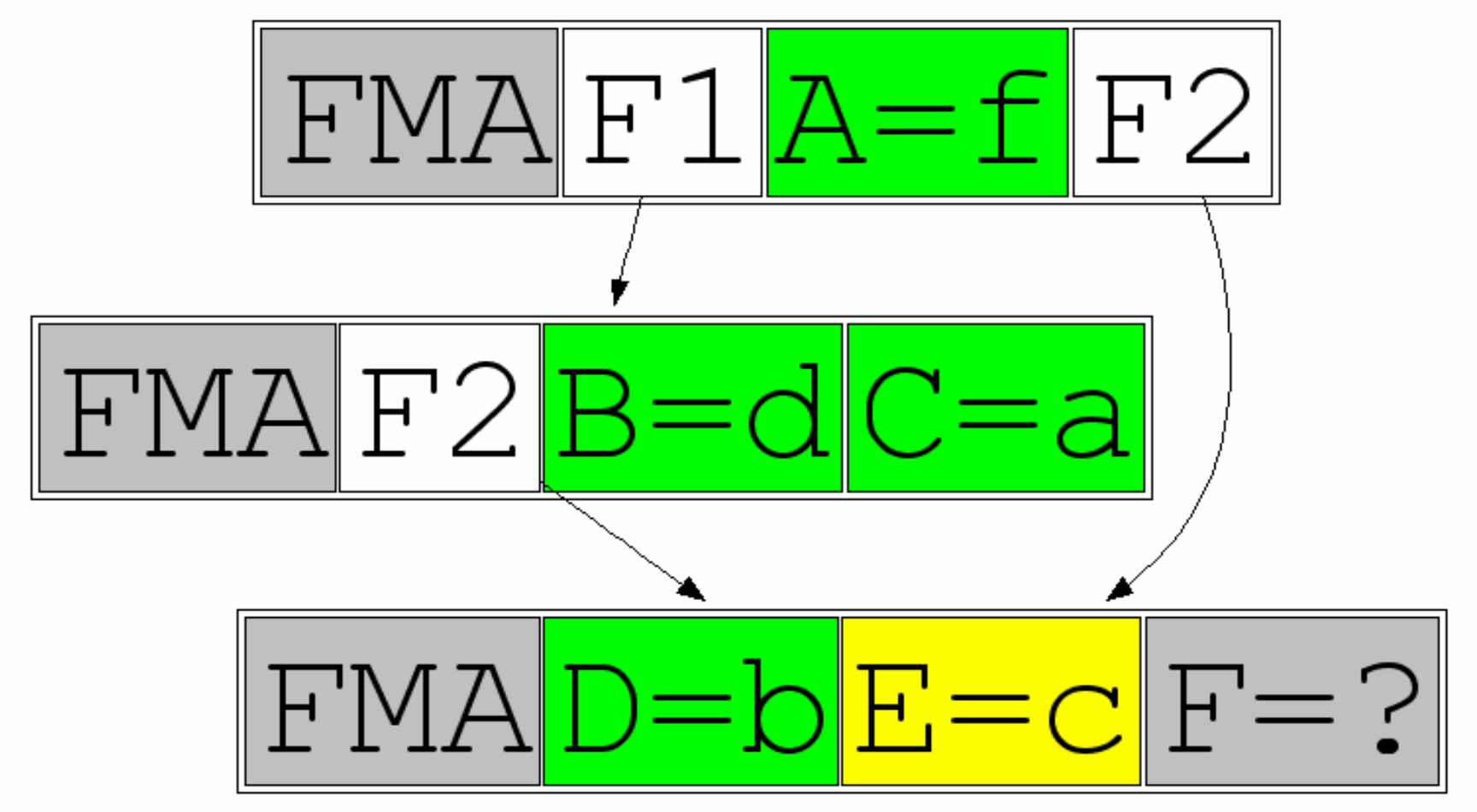
Formal: +ABDE+ABF+AC+DE+F

Actual: +abdf+cdf+ab-ef+c

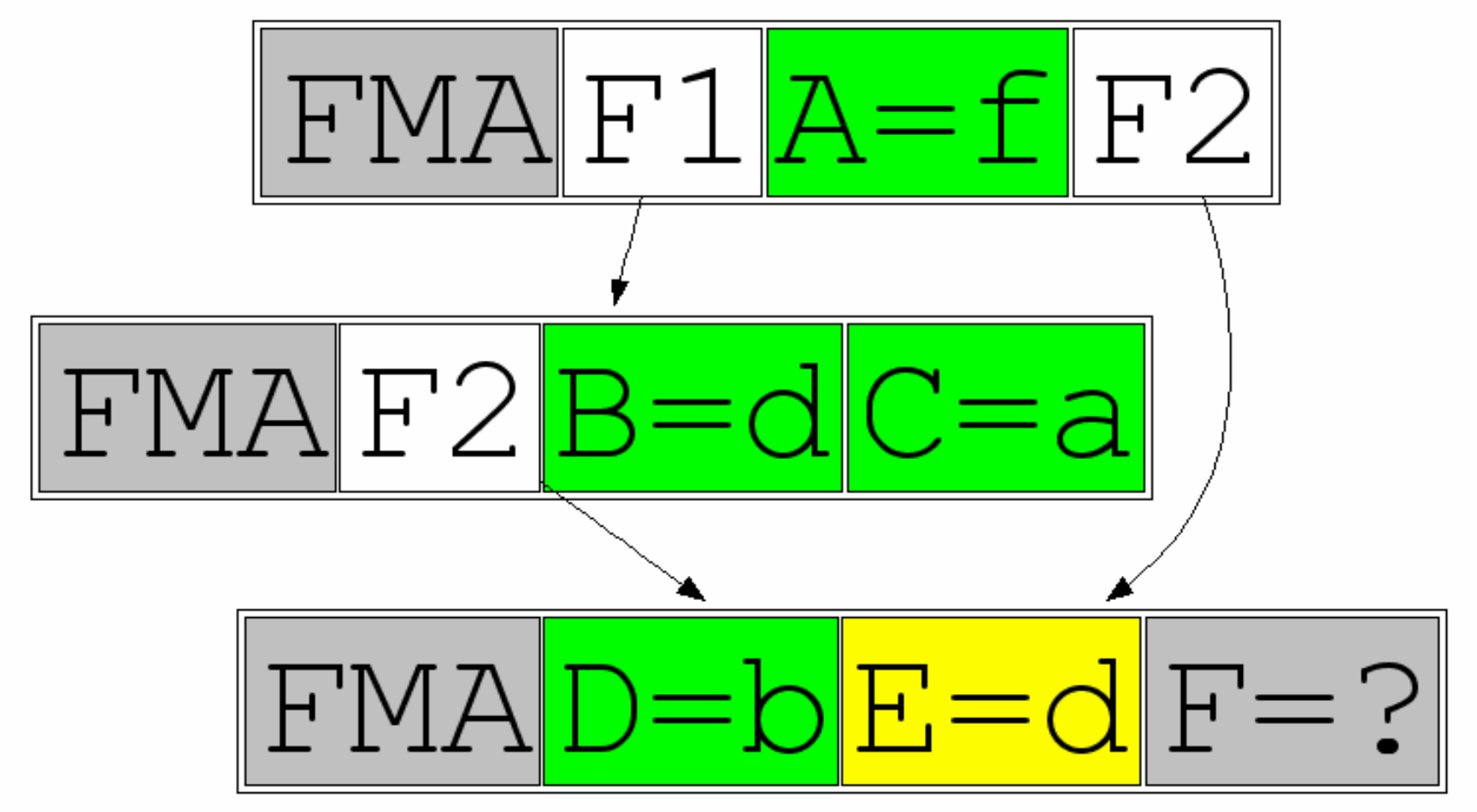


Formal: +ABDE+ABF+AC+DE+F

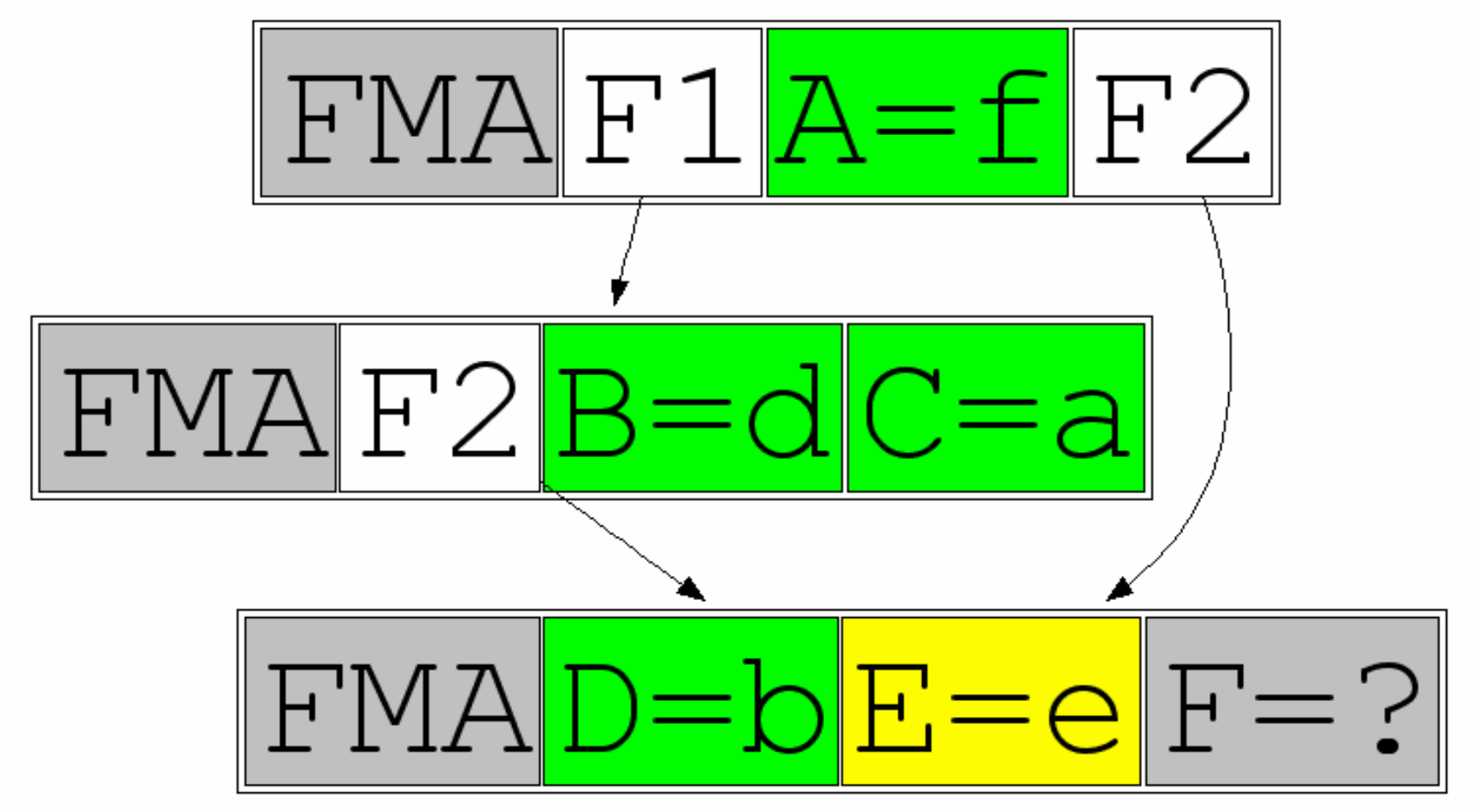
Actual: +abdf+cdf+ab-ef+c



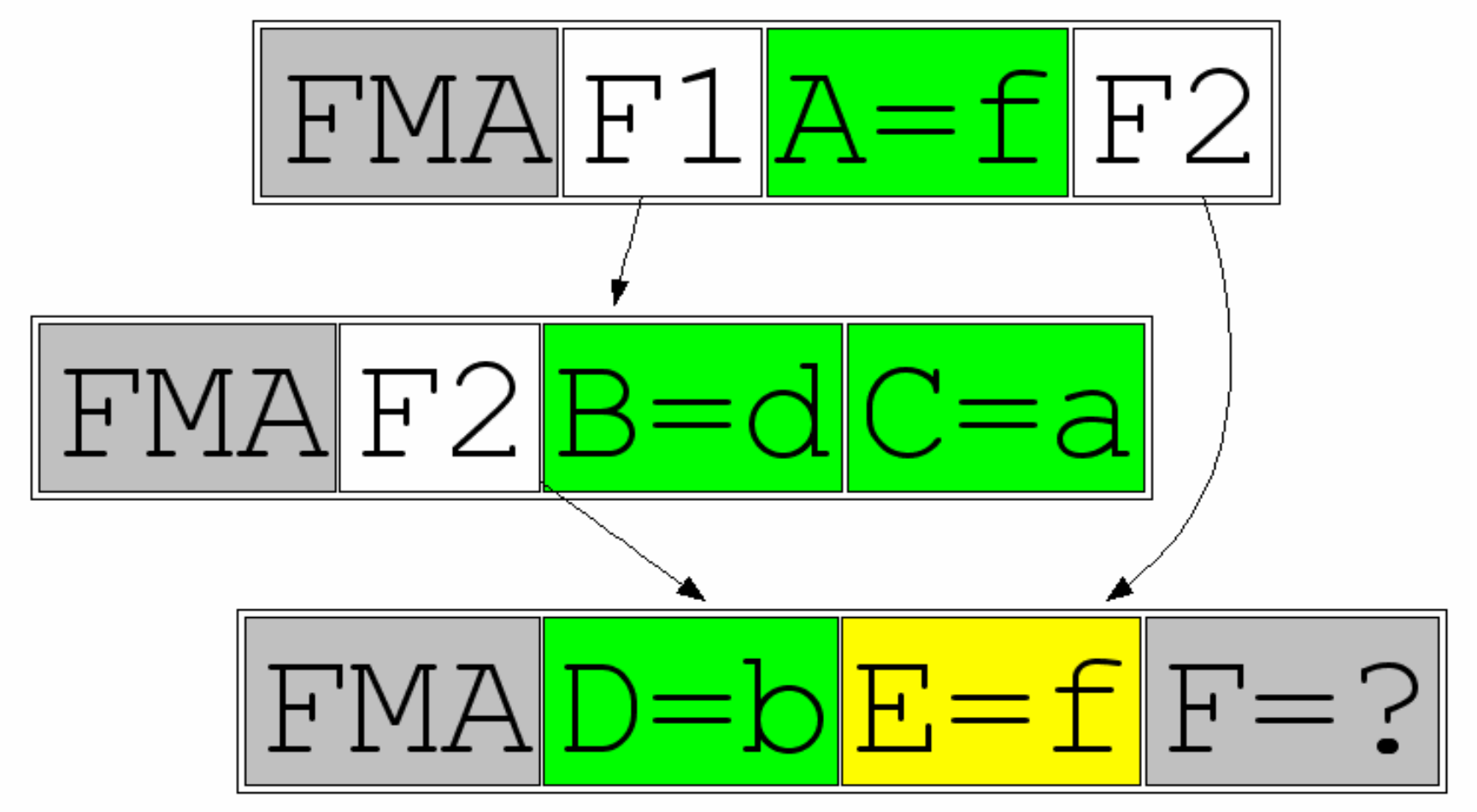
Formal: +ABDE+ABF+AC+DE+F
Actual: +abdf+cdf+ab-ef+c



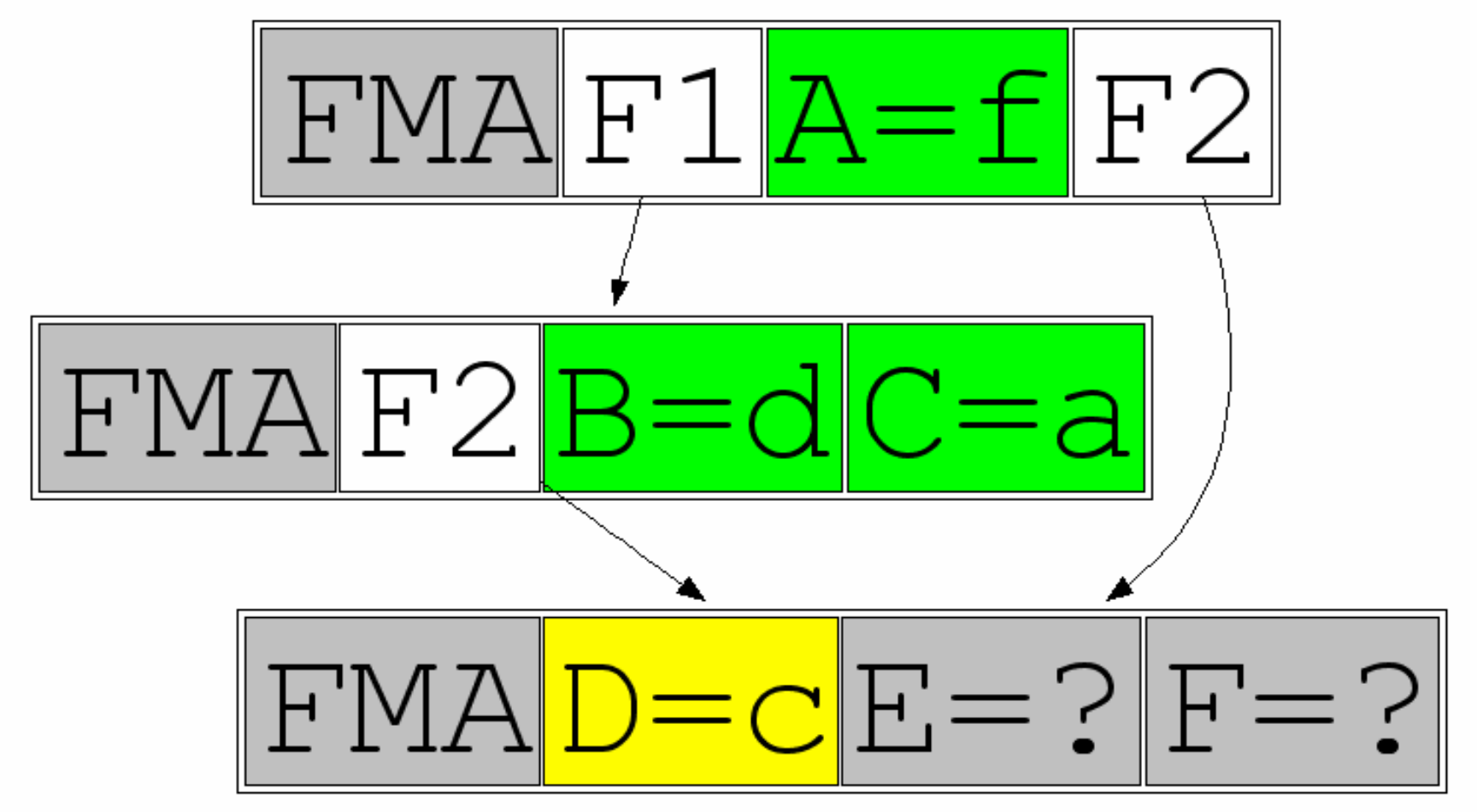
Formal: +ABDE+ABF+AC+DE+F
Actual: +abdf+cdf+ab-ef+c



Formal: +ABDE+ABF+AC+DE+F
Actual: +abdf+cdf+ab-ef+c

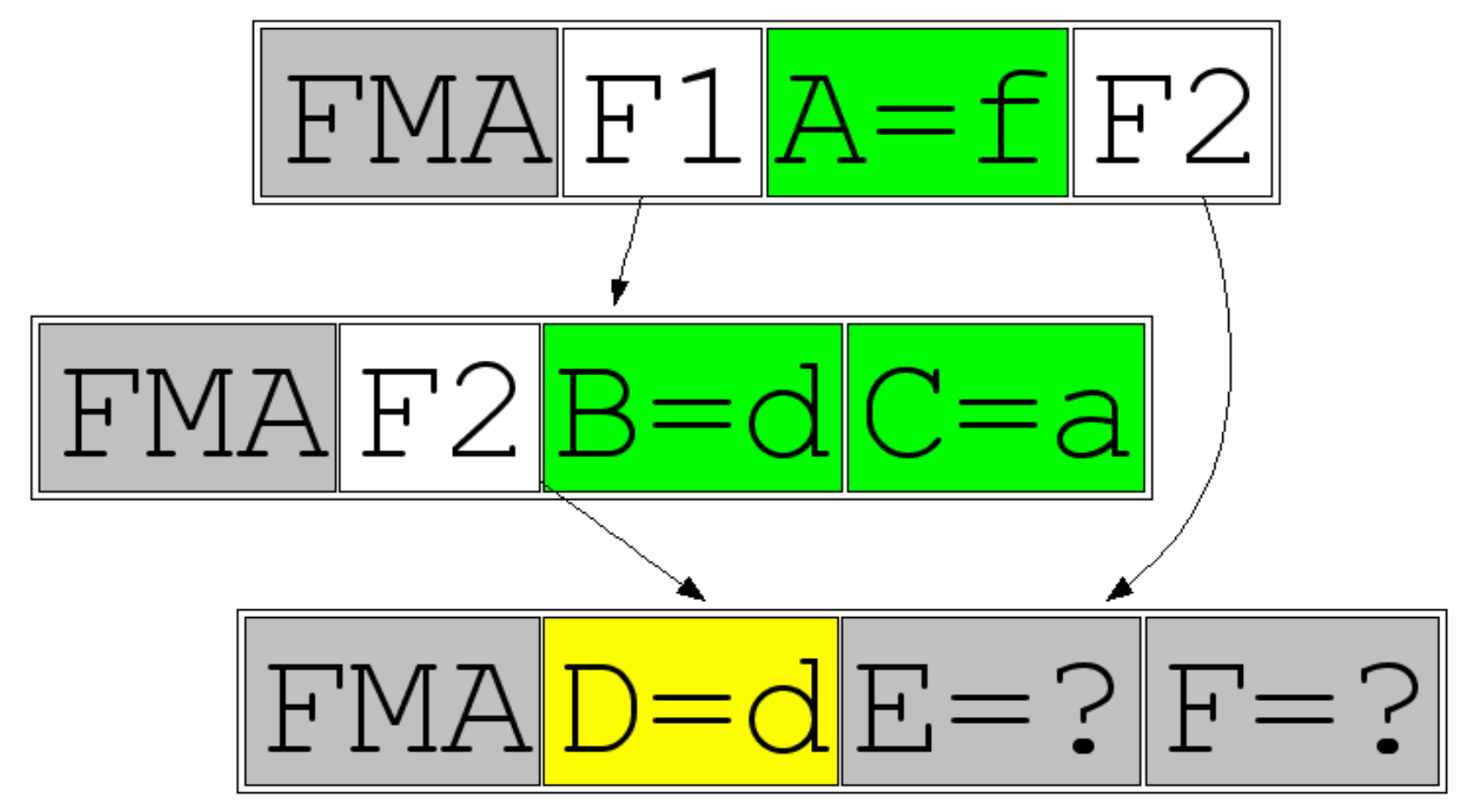


Formal: +ABDE+ABF+AC+DE+F
Actual: +abdf+cdf+ab-ef+c



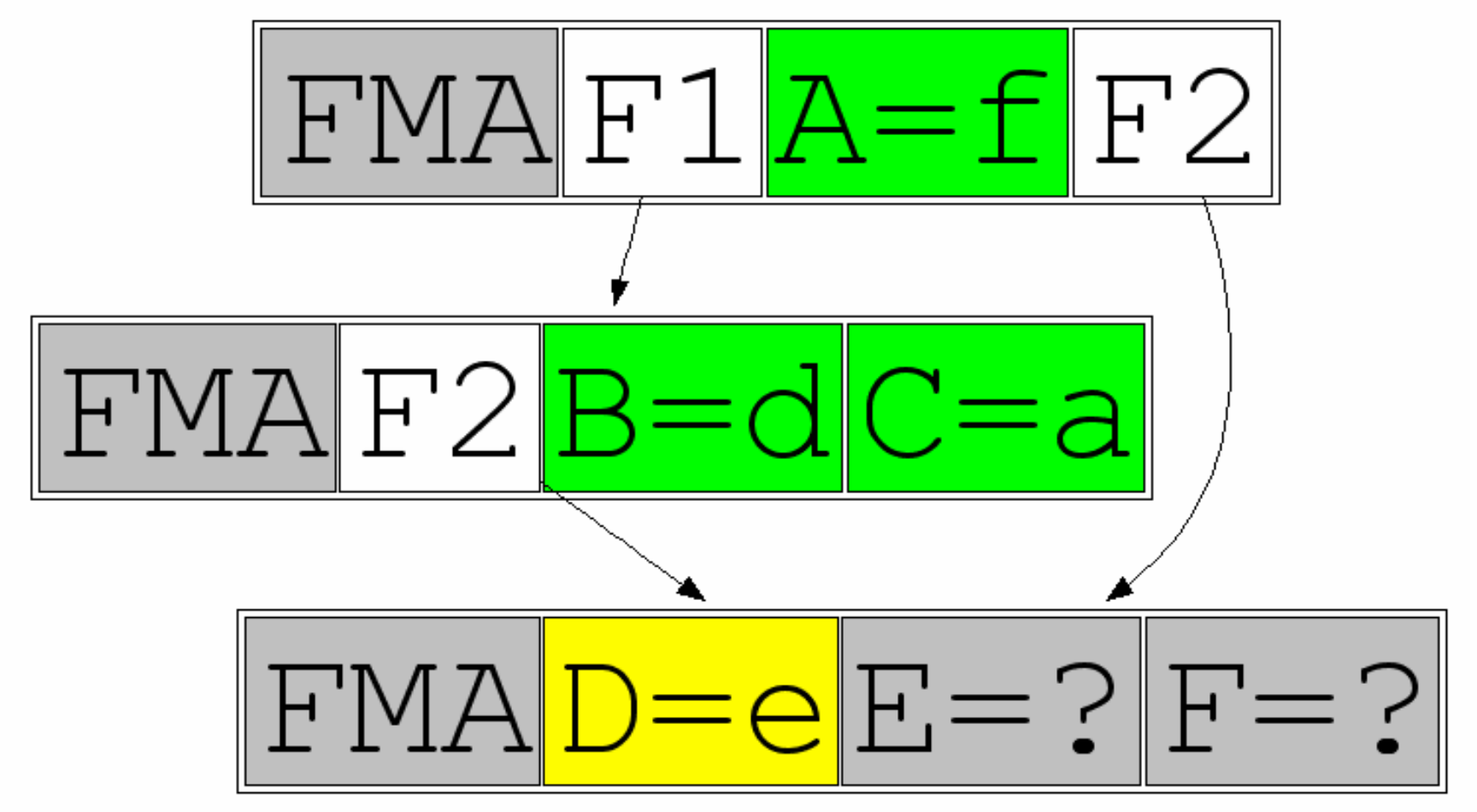
Formal: +ABDE+ABF+AC+DE+F

Actual: +abdf+cdf+ab-ef+c

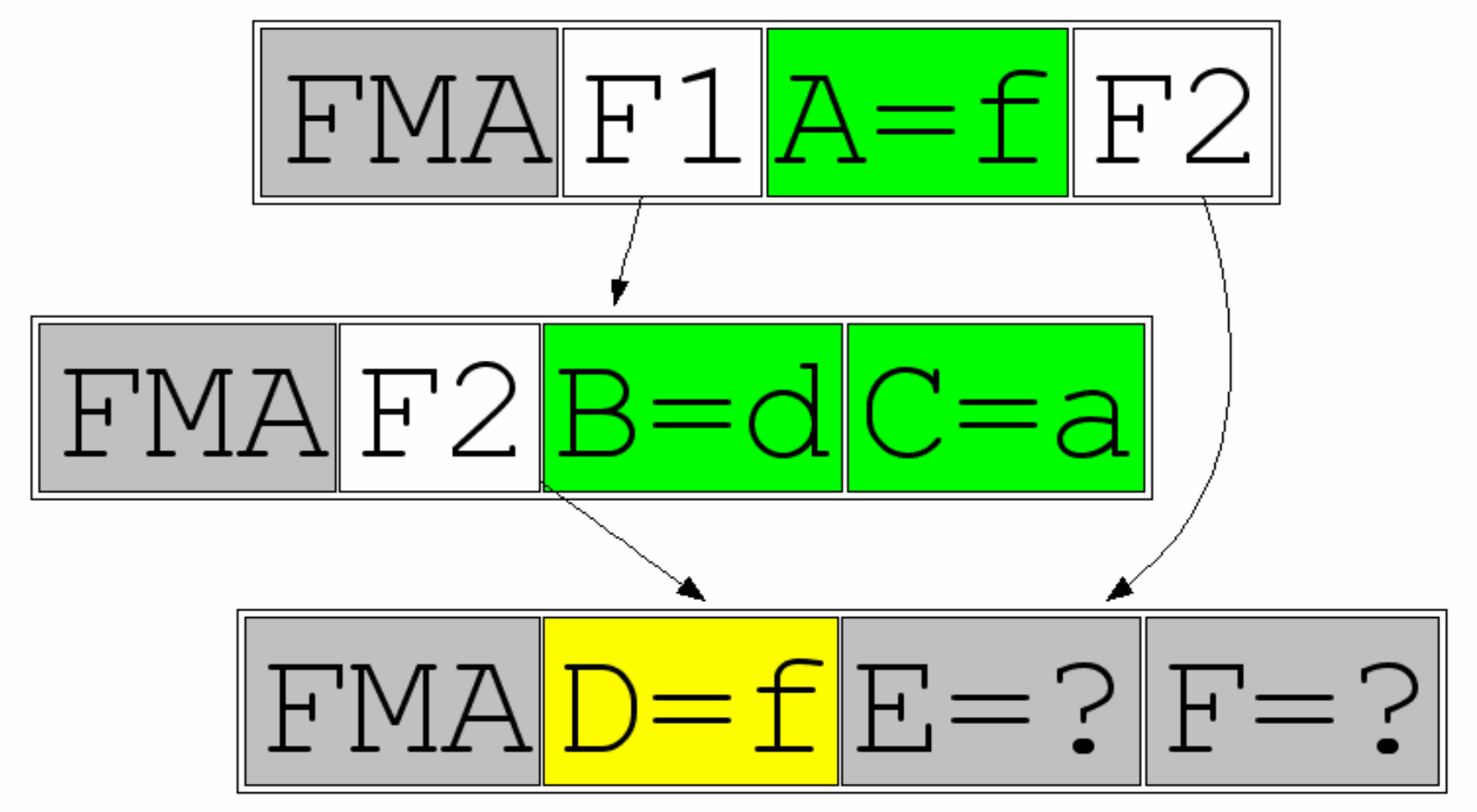


Formal: +ABDE+ABF+AC+DE+F

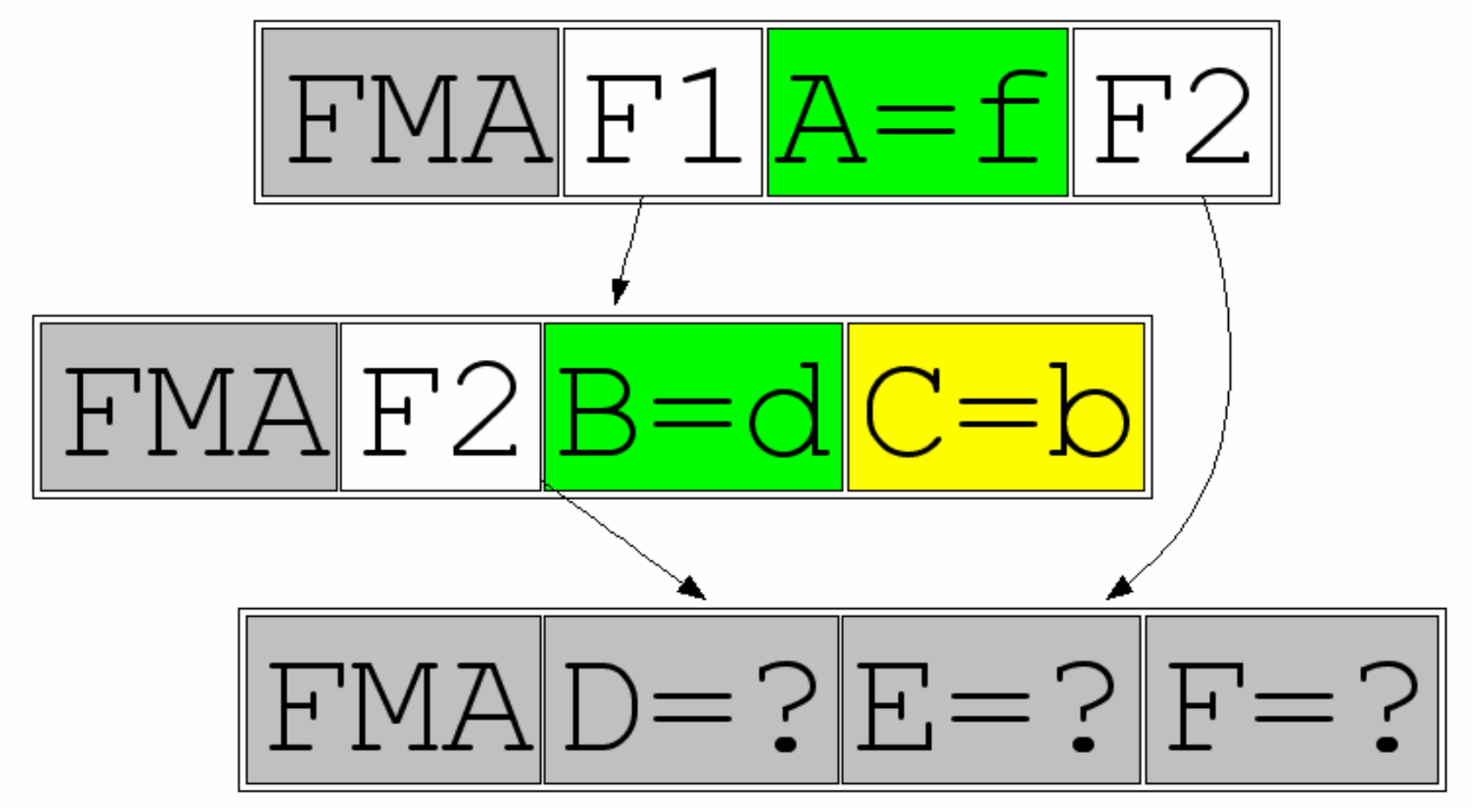
Actual: +abdf+cdf+ab-ef+c



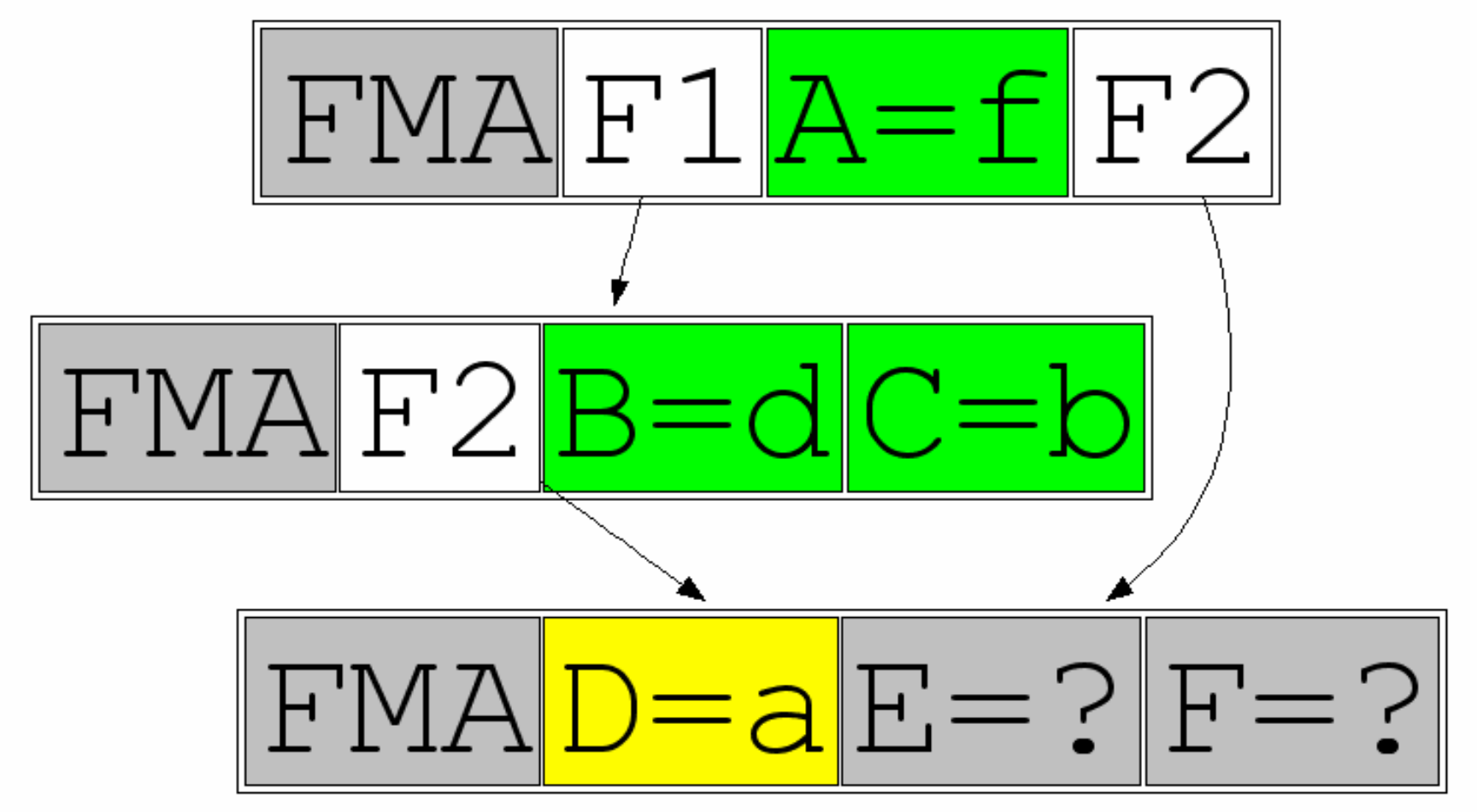
Formal: +ABDE+ABF+AC+DE+F
Actual: +abdf+cdf+ab-ef+c



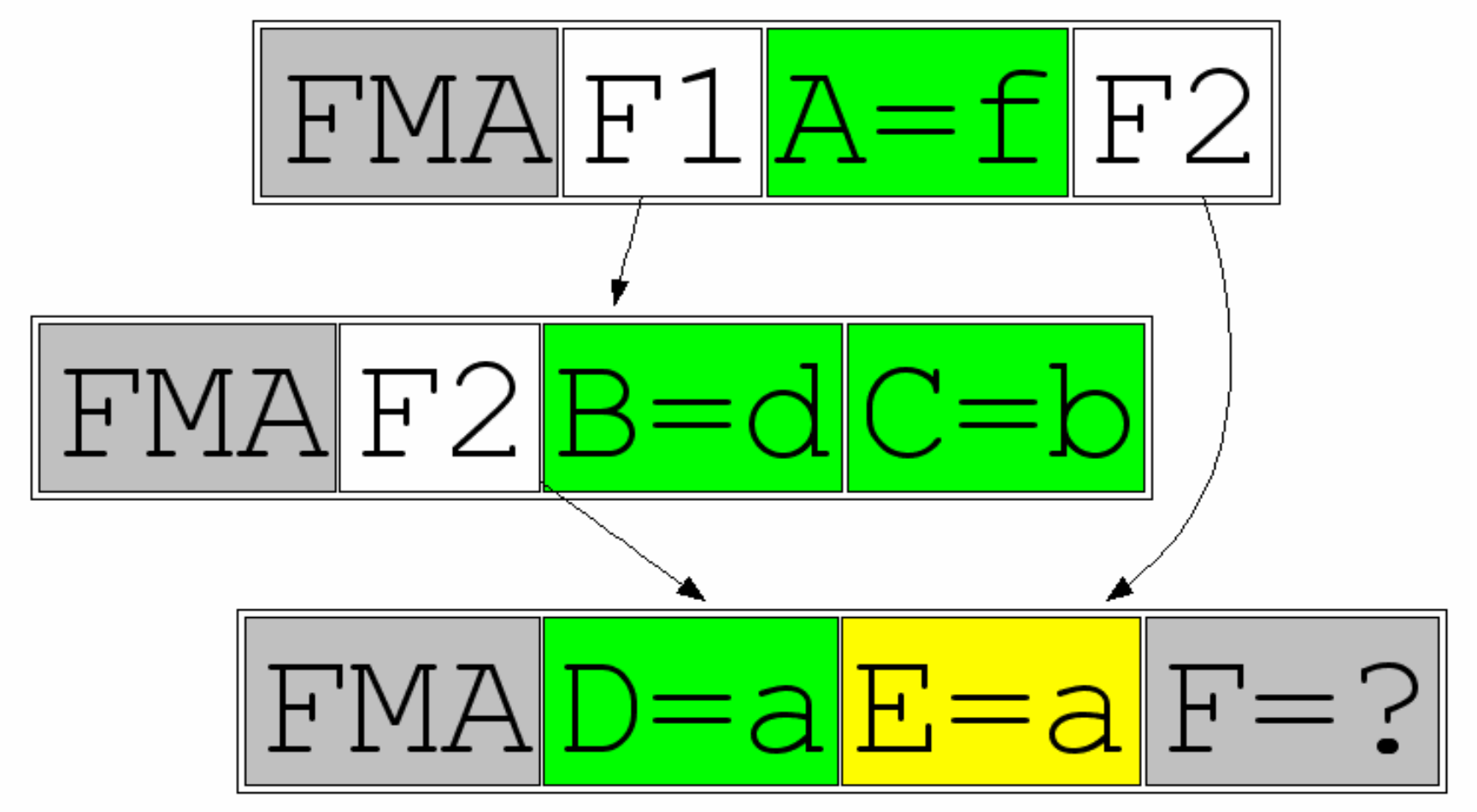
Formal: +ABDE+ABF+AC+DE+F
Actual: +abdf+cdf+ab-ef+c



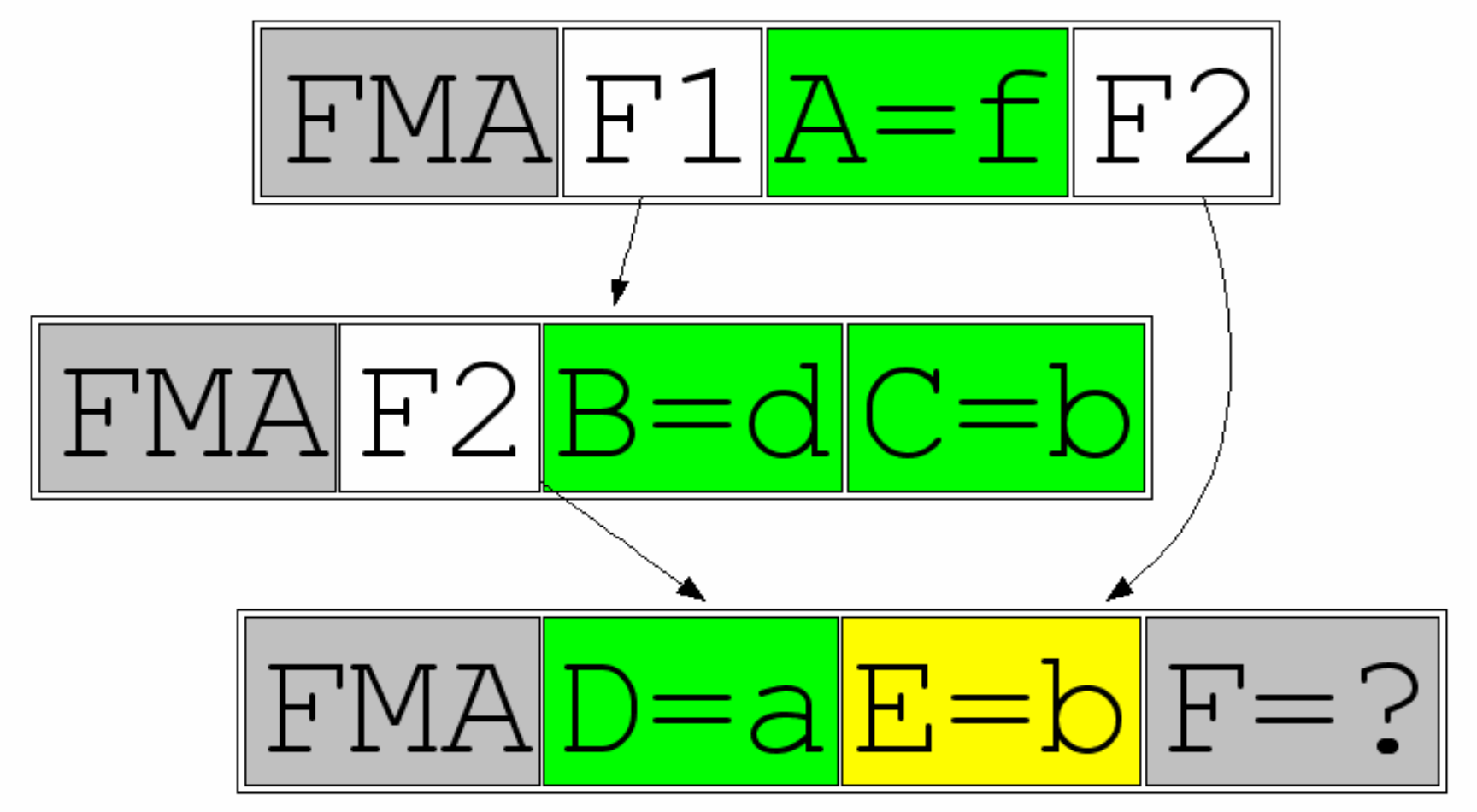
Formal: +ABDE+ABF+AC+DE+F
Actual: +abdf+cdf+ab-ef+c



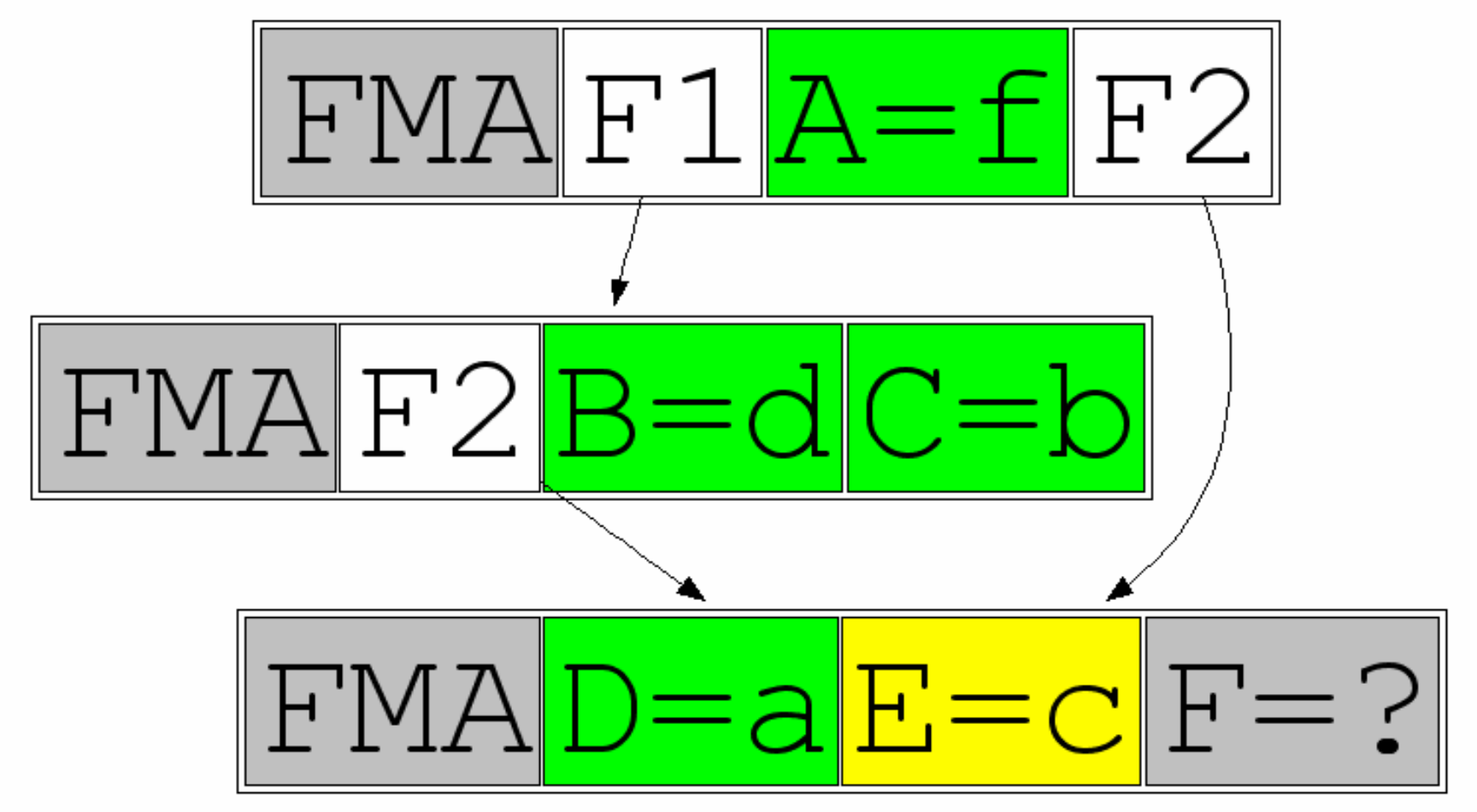
Formal: +ABDE+ABF+AC+DE+F
Actual: +abdf+cdf+ab-ef+c



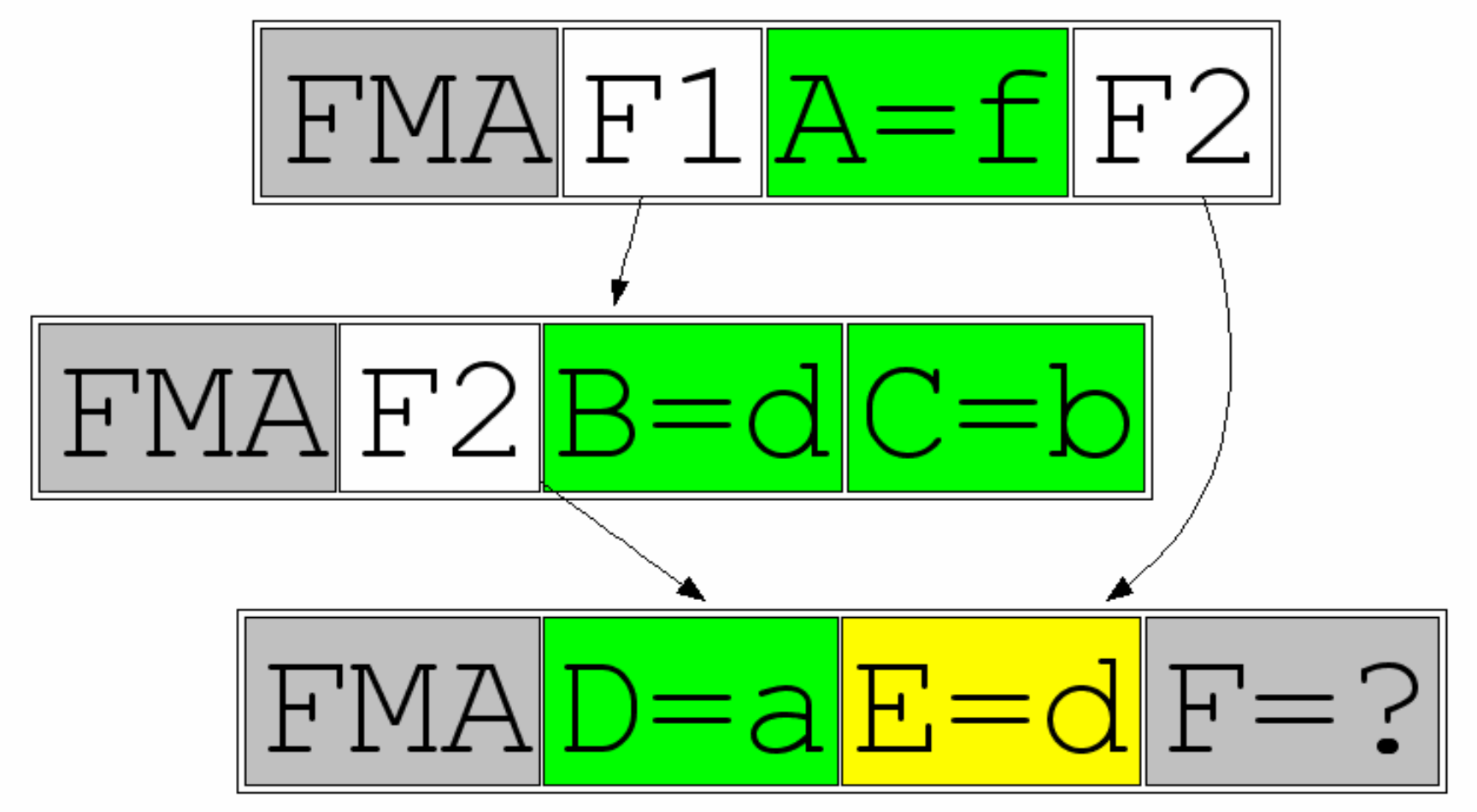
Formal: +ABDE+ABF+AC+DE+F
Actual: +abdf+cdf+ab-ef+c



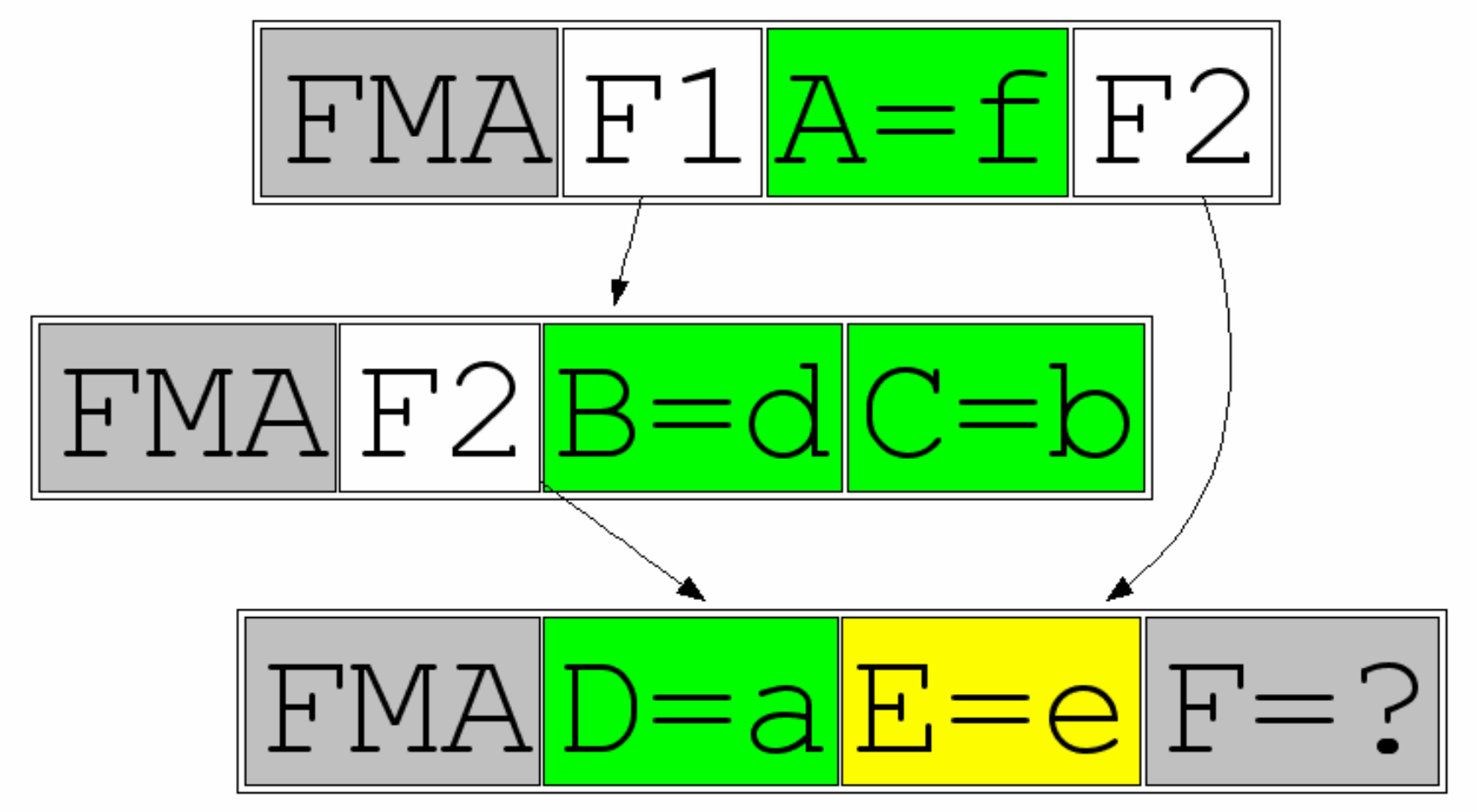
Formal: +ABDE+ABF+AC+DE+F
Actual: +abdf+cdf+ab-ef+c



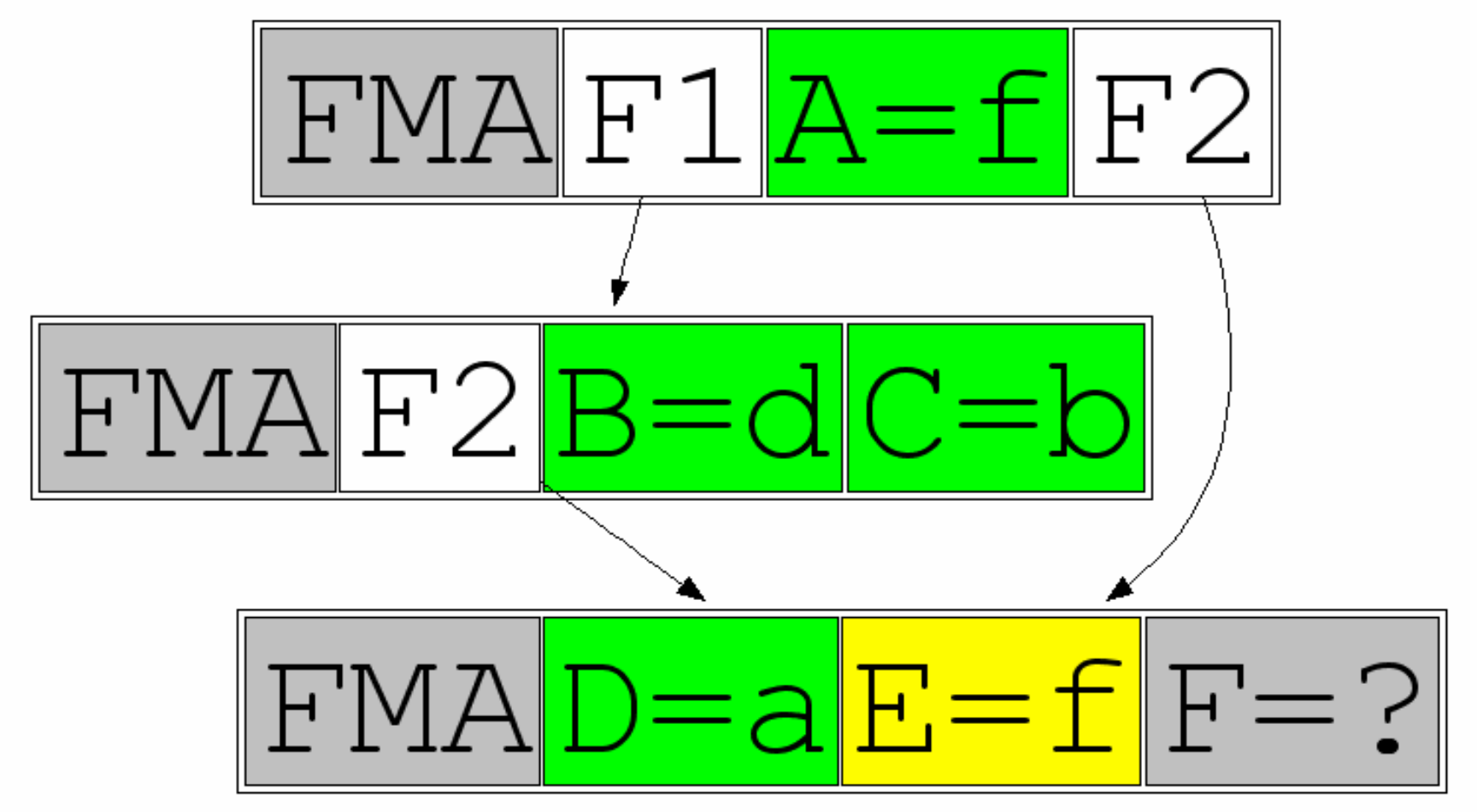
Formal: +ABDE+ABF+AC+DE+F
Actual: +abdf+cdf+ab-ef+c



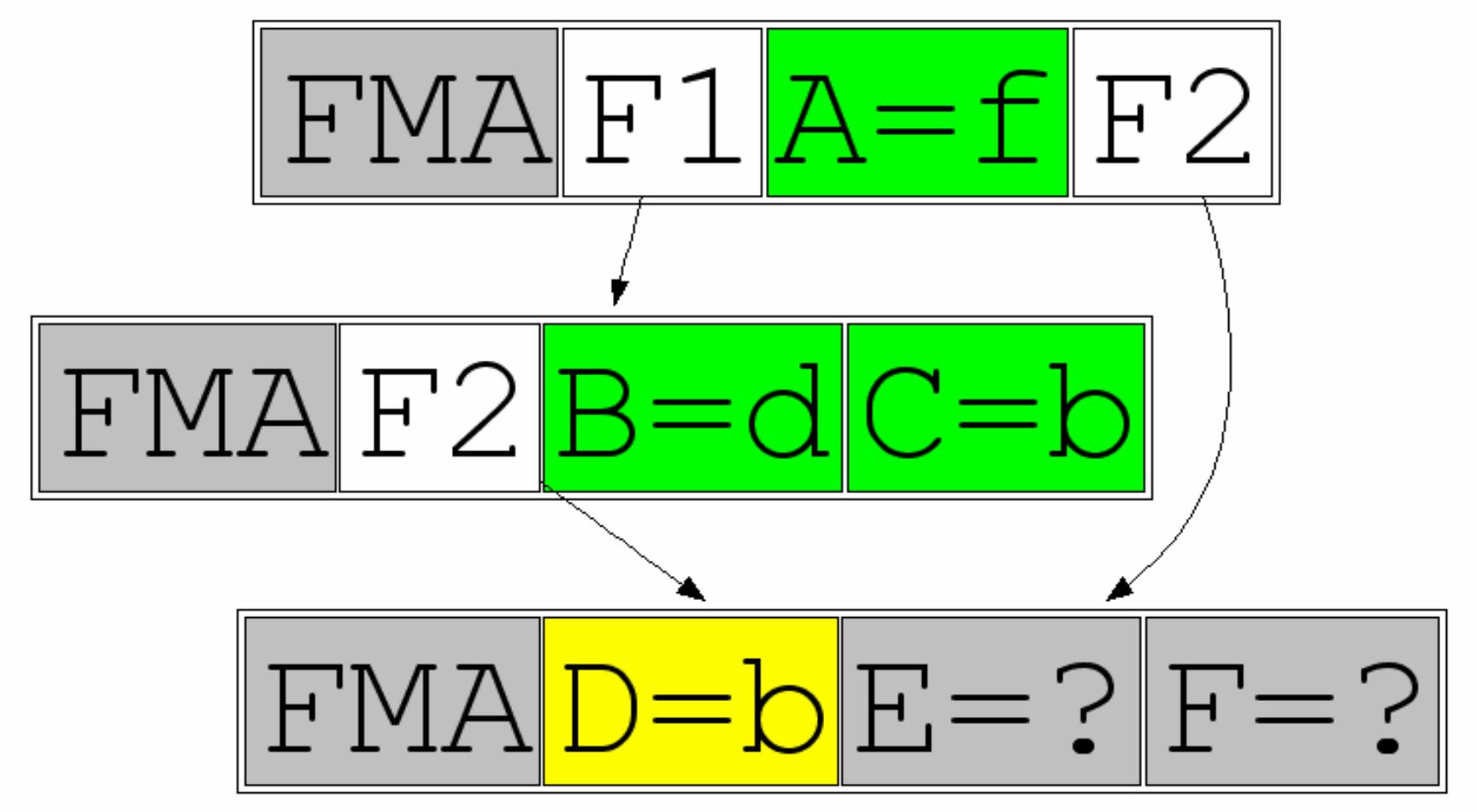
Formal: +ABDE+ABF+AC+DE+F
Actual: +abdf+cdf+ab-ef+c



Formal: +ABDE+ABF+AC+DE+F
Actual: +abdf+cdf+ab-ef+c

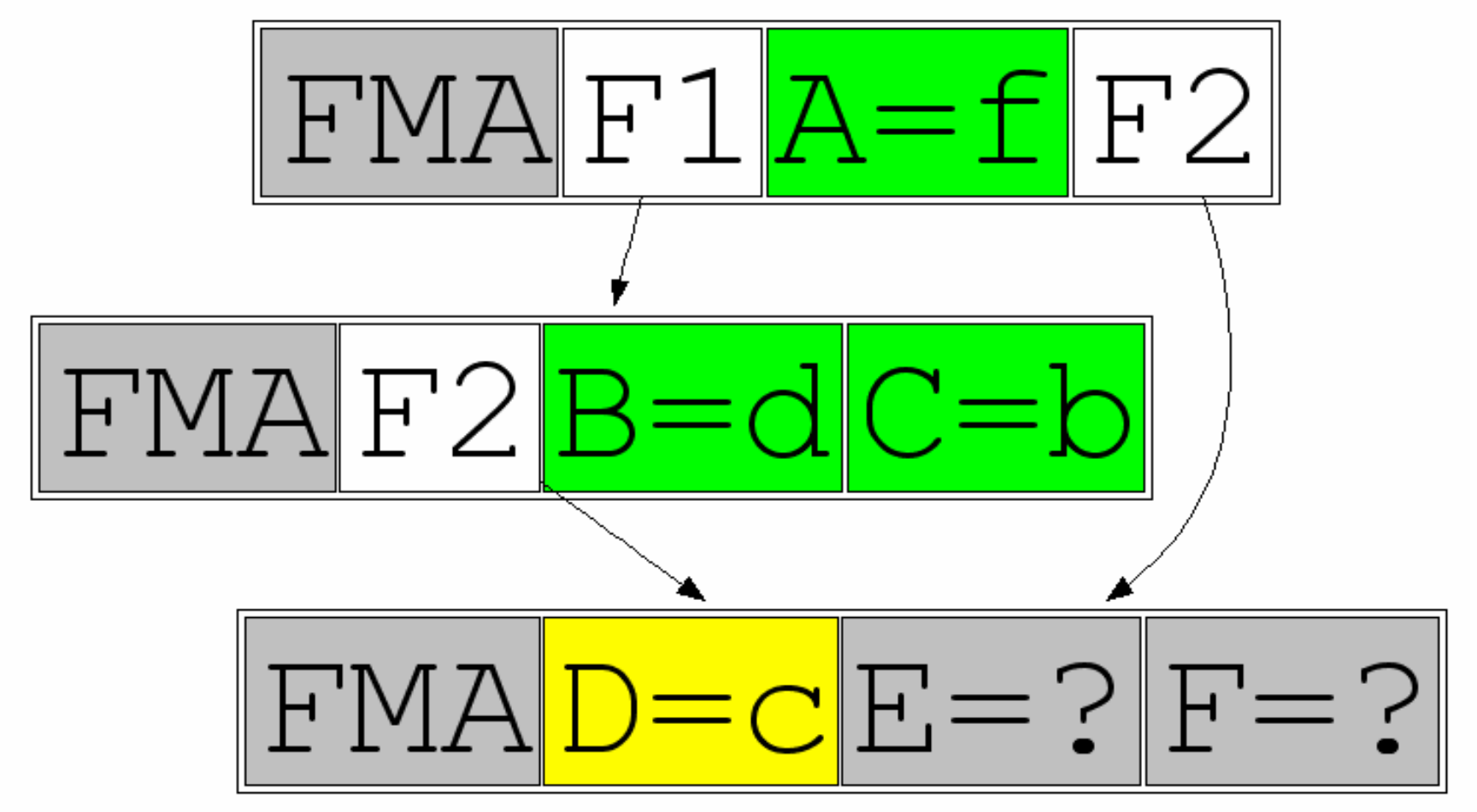


Formal: +ABDE+ABF+AC+DE+F
Actual: +abdf+cdf+ab-ef+c

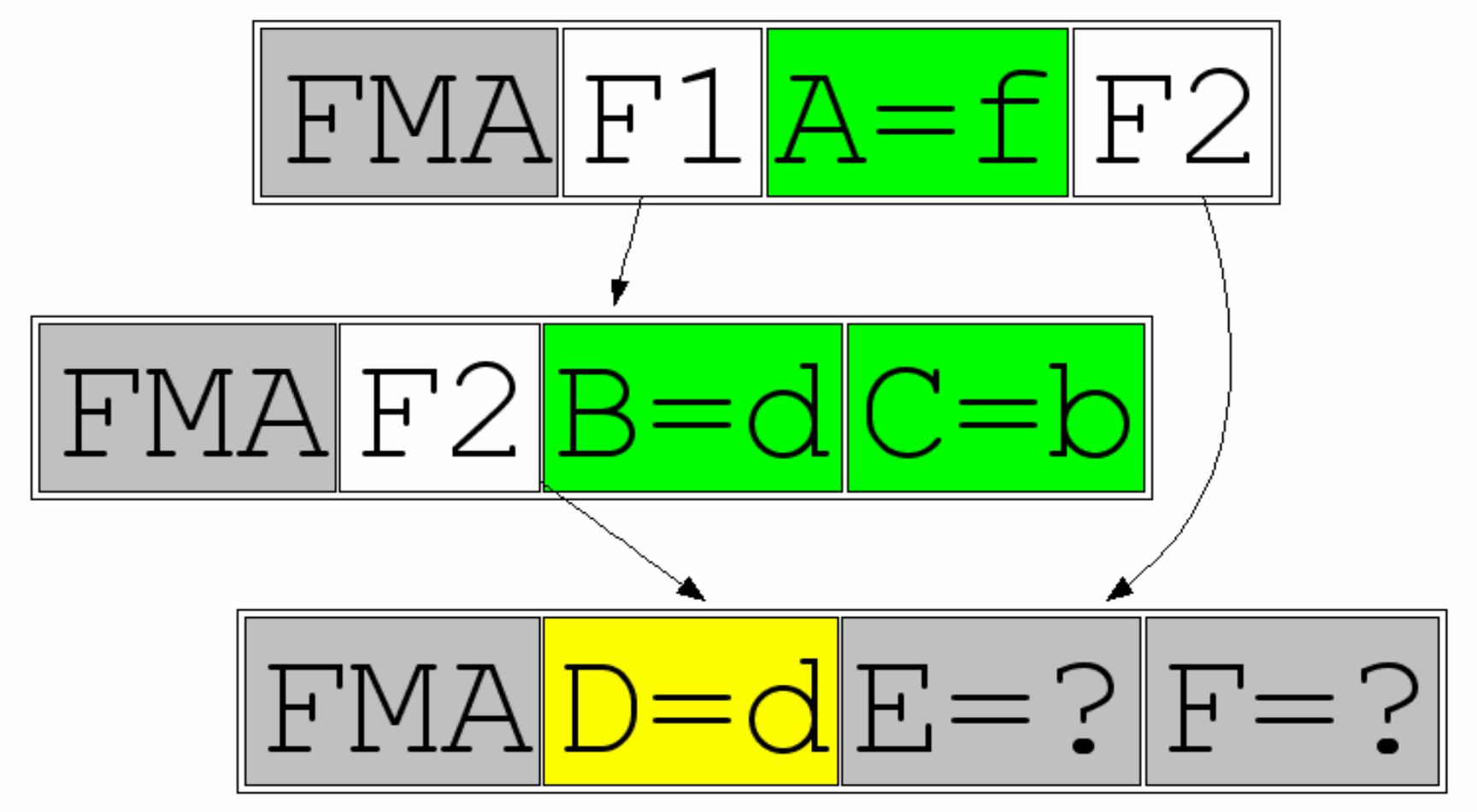


Formal: +ABDE+ABF+AC+DE+F

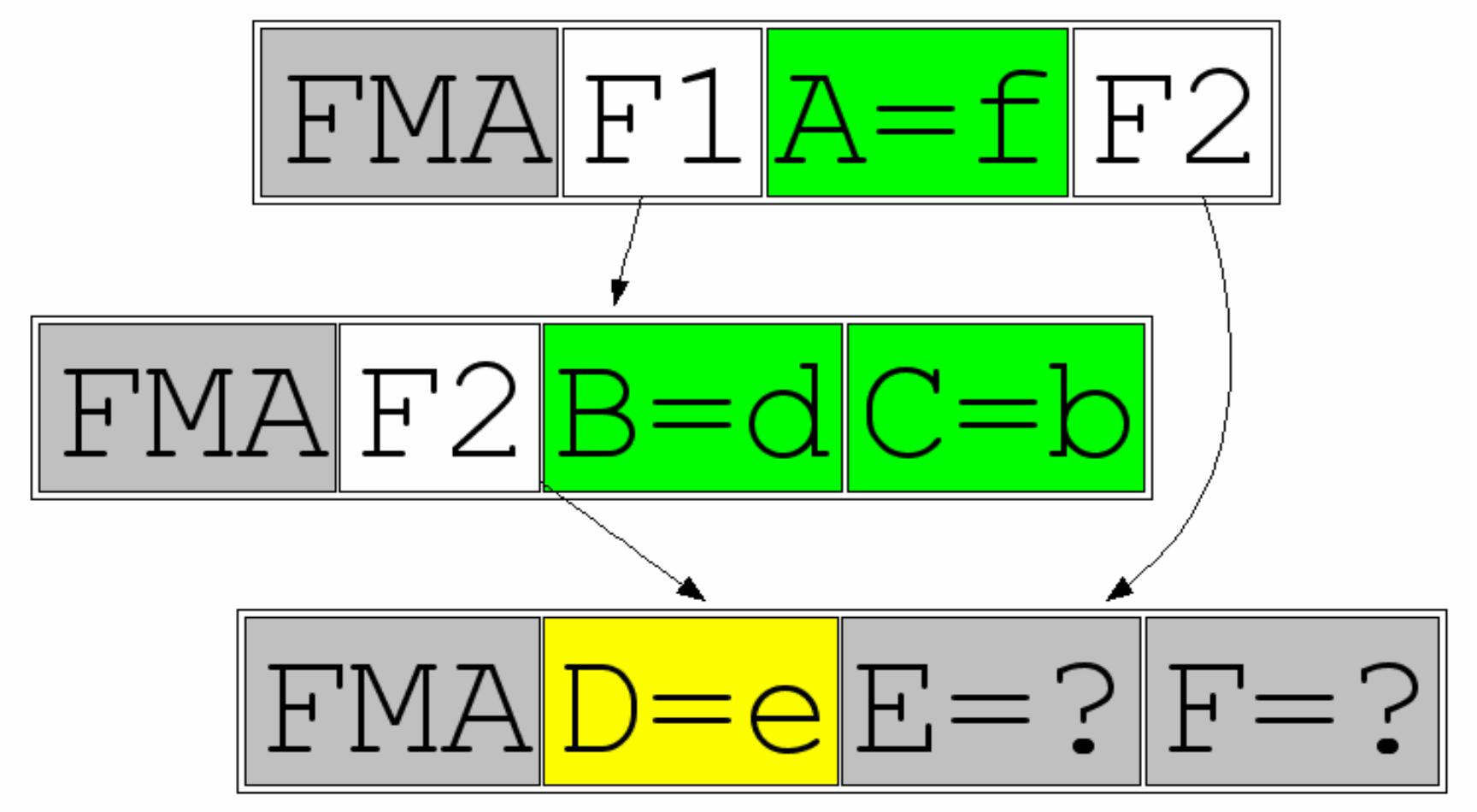
Actual: +abdf+cdf+ab-ef+c



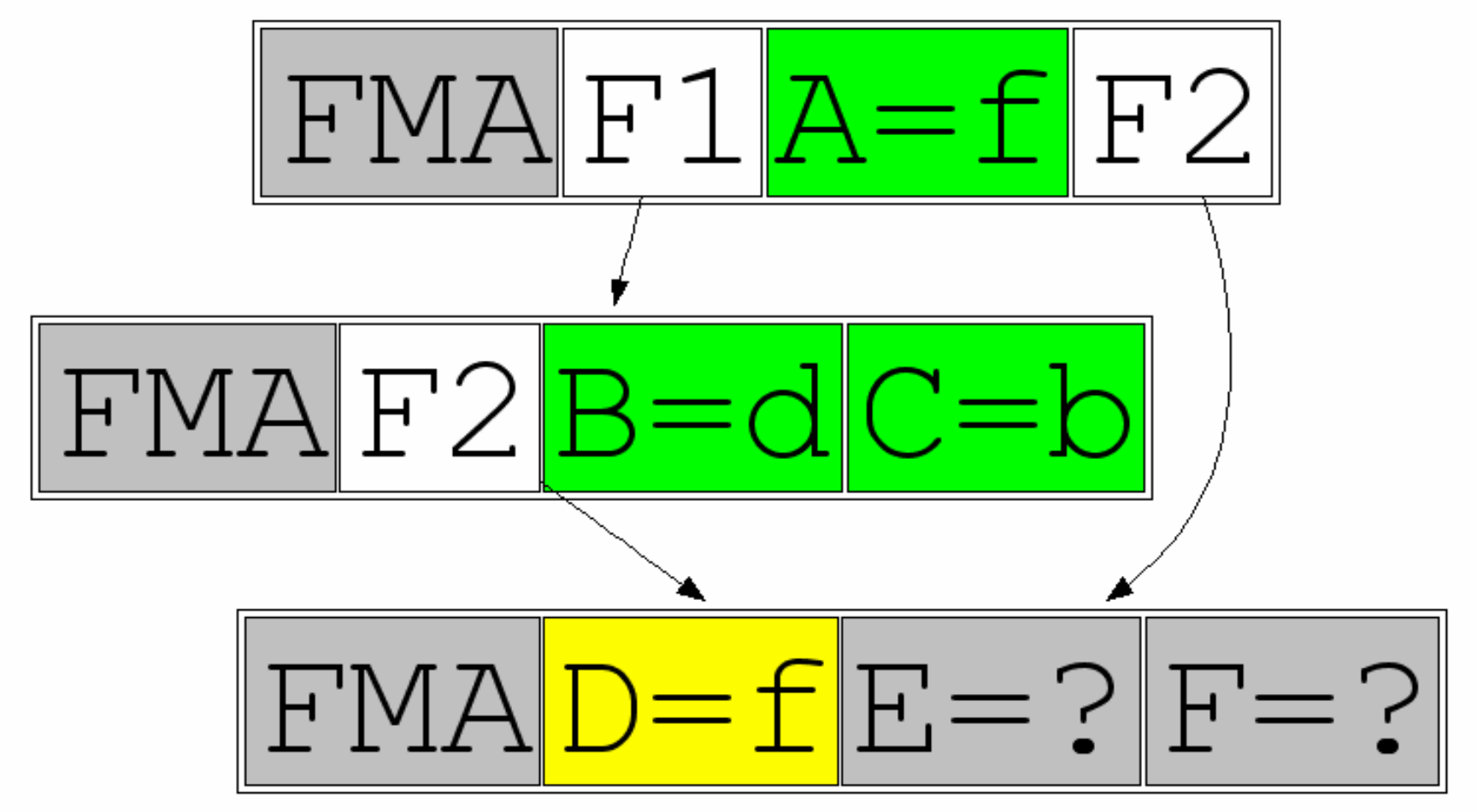
Formal: +ABDE+ABF+AC+DE+F
Actual: +abdf+cdf+ab-ef+c



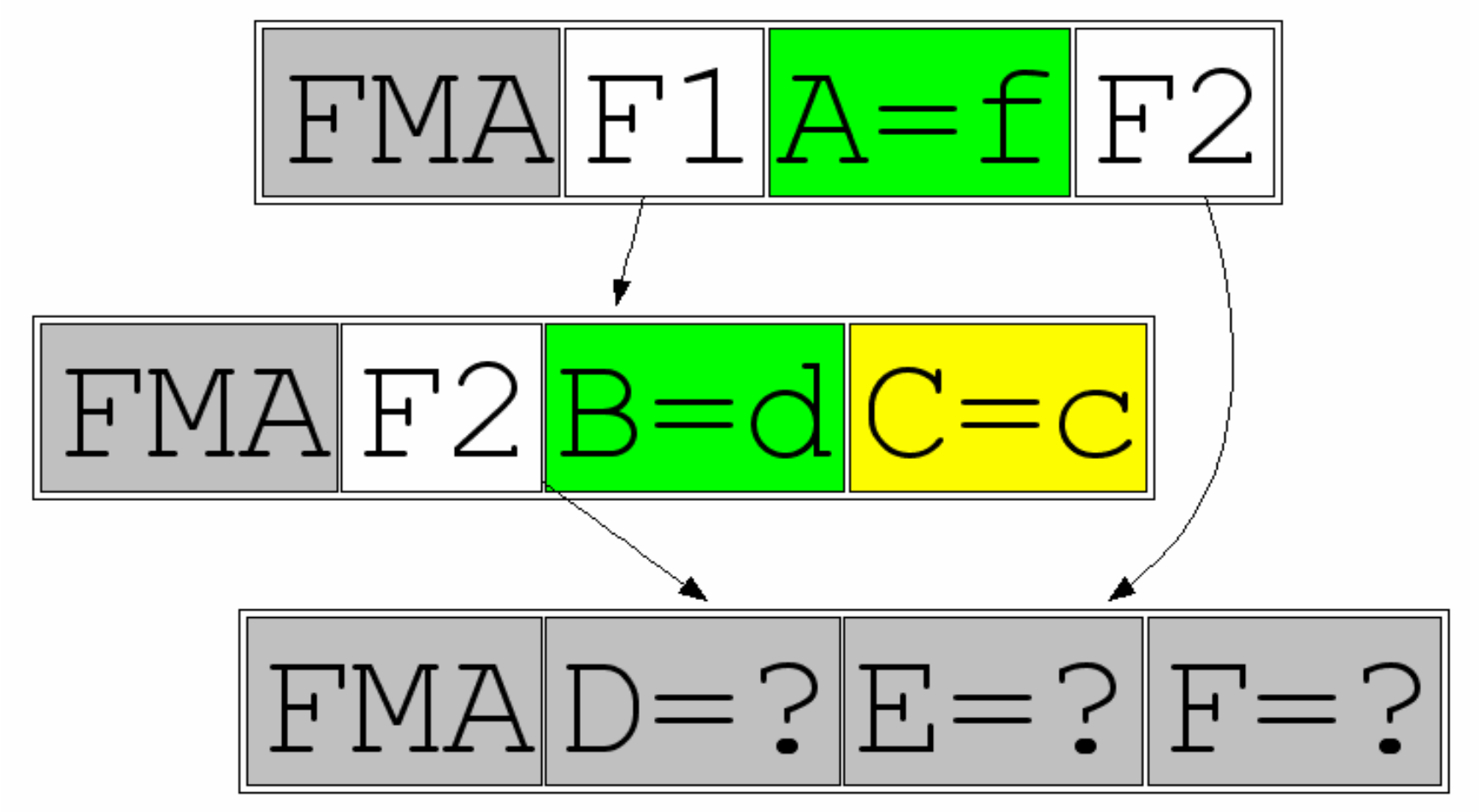
Formal: +ABDE+ABF+AC+DE+F
Actual: +abdf+cdf+ab-ef+c



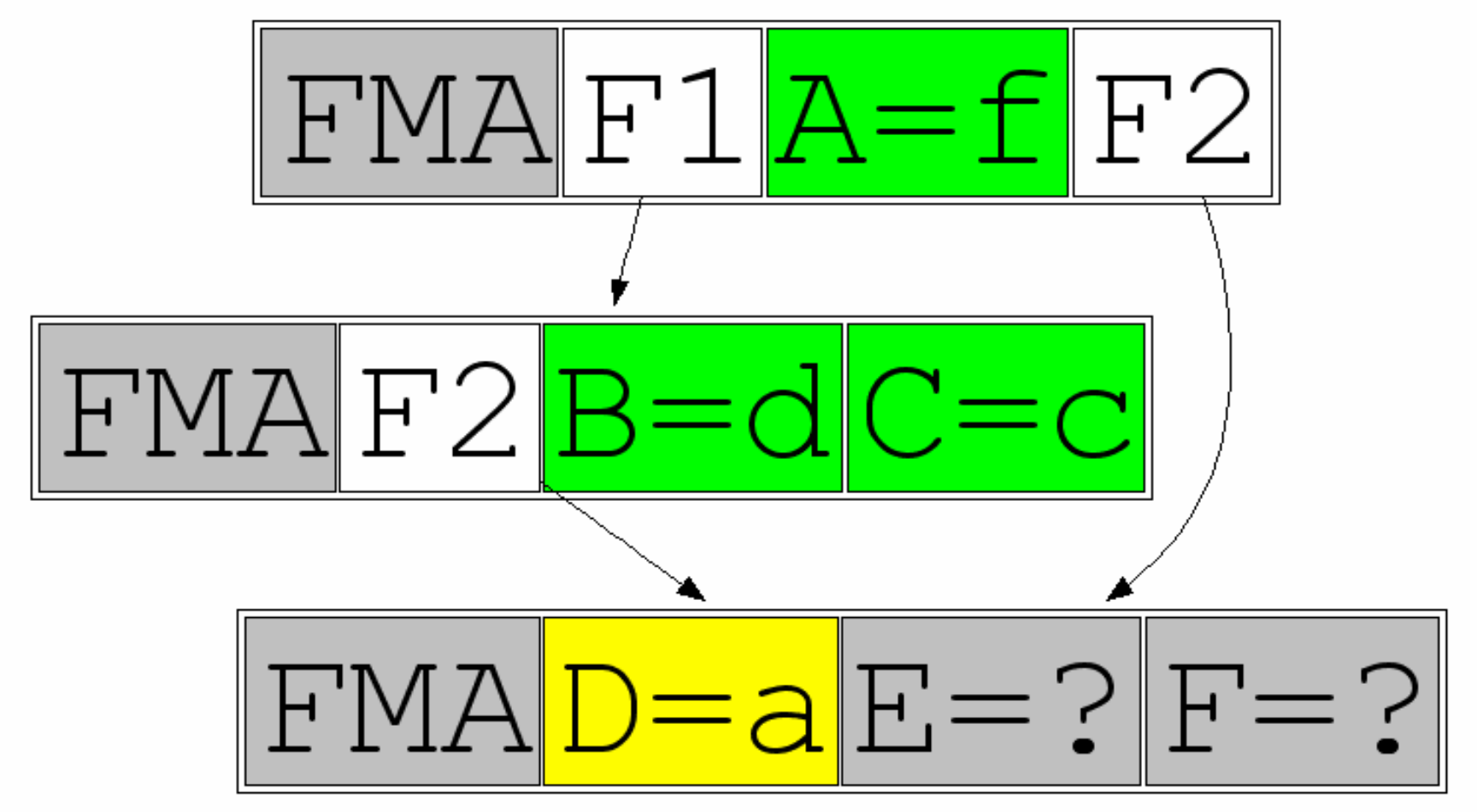
Formal: +ABDE+ABF+AC+DE+F
Actual: +abdf+cdf+ab-ef+c



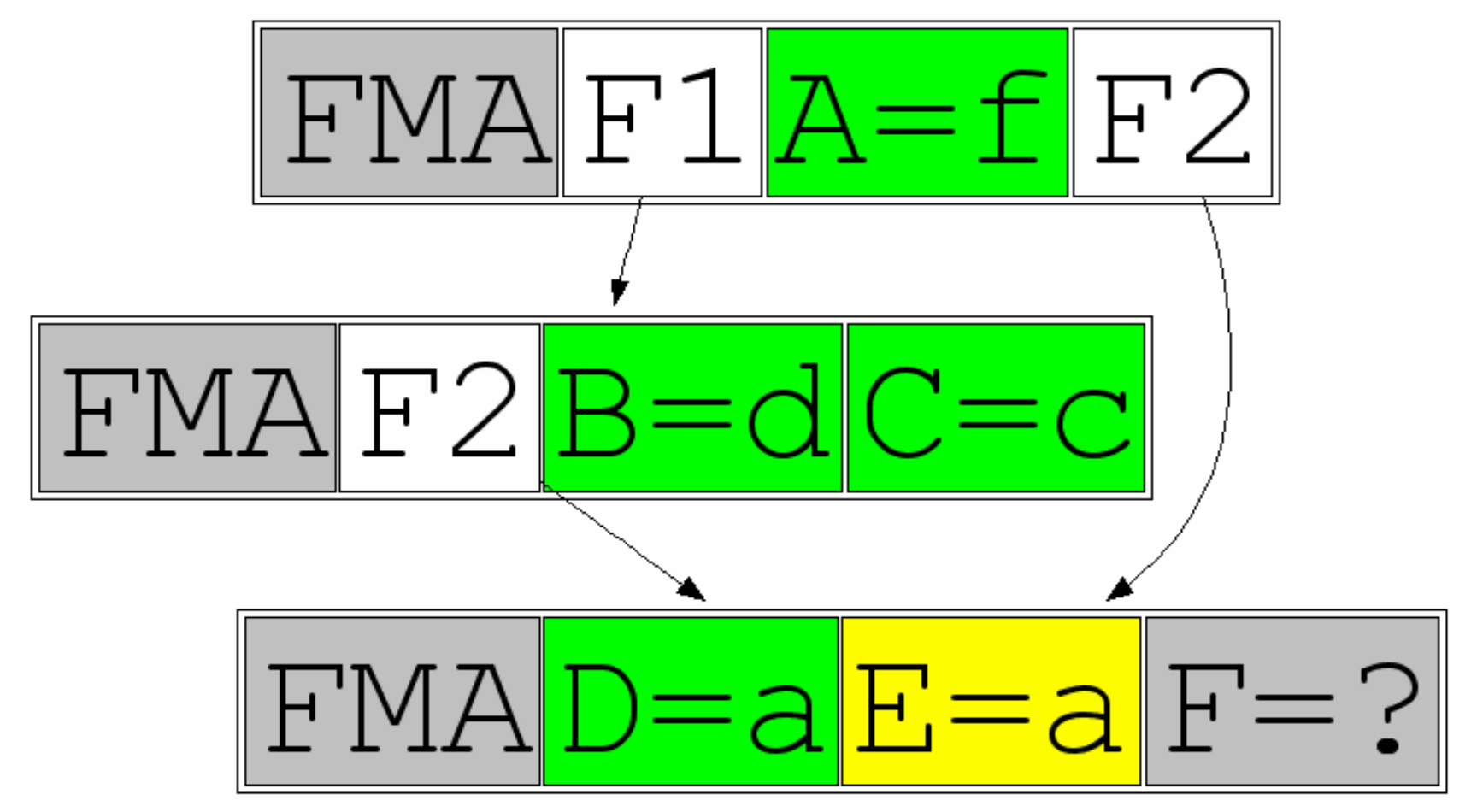
Formal: +ABDE+ABF+AC+DE+F
Actual: +abdf+cdf+ab-ef+c



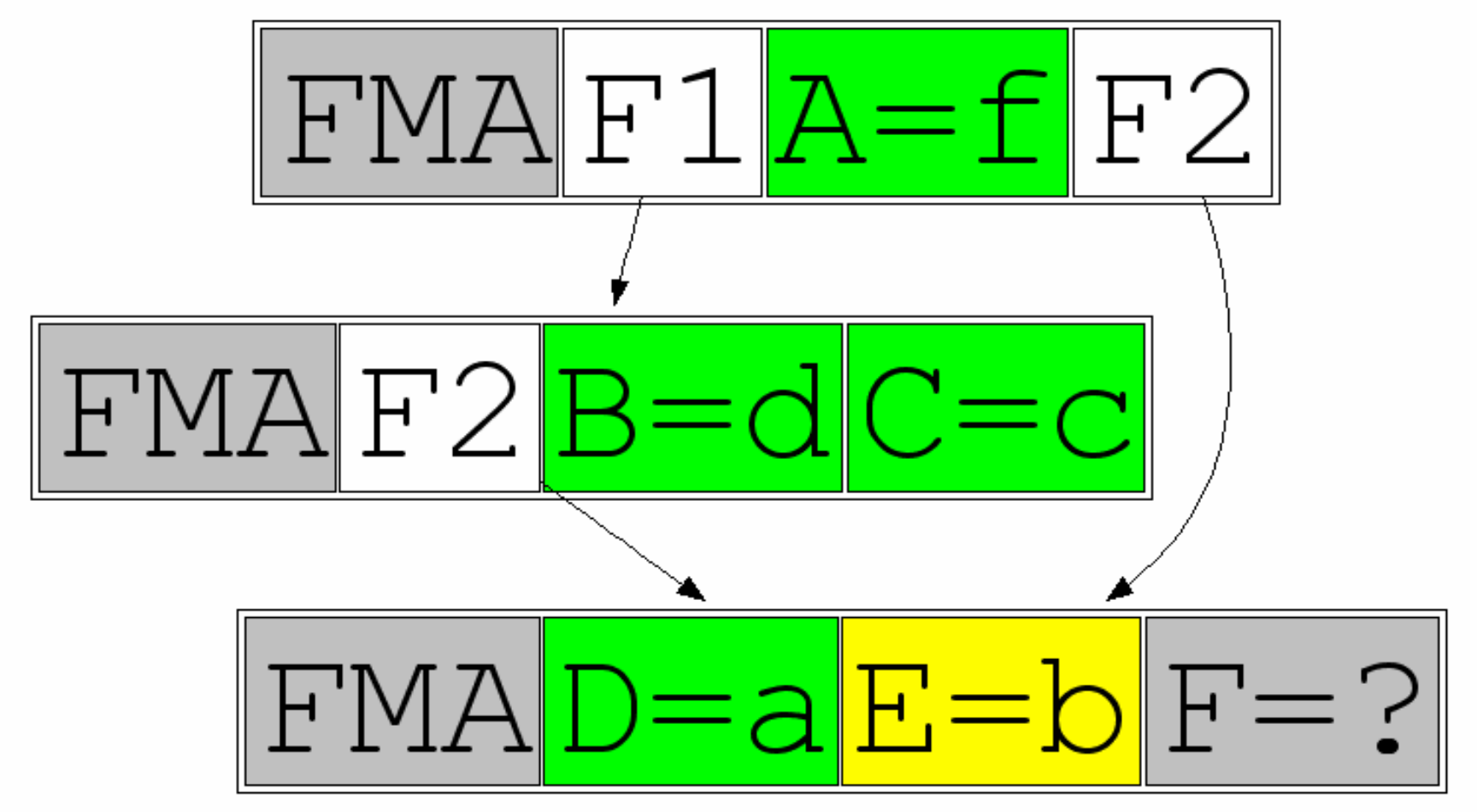
Formal: +ABDE+ABF+AC+DE+F
Actual: +abdf+cdf+ab-ef+c



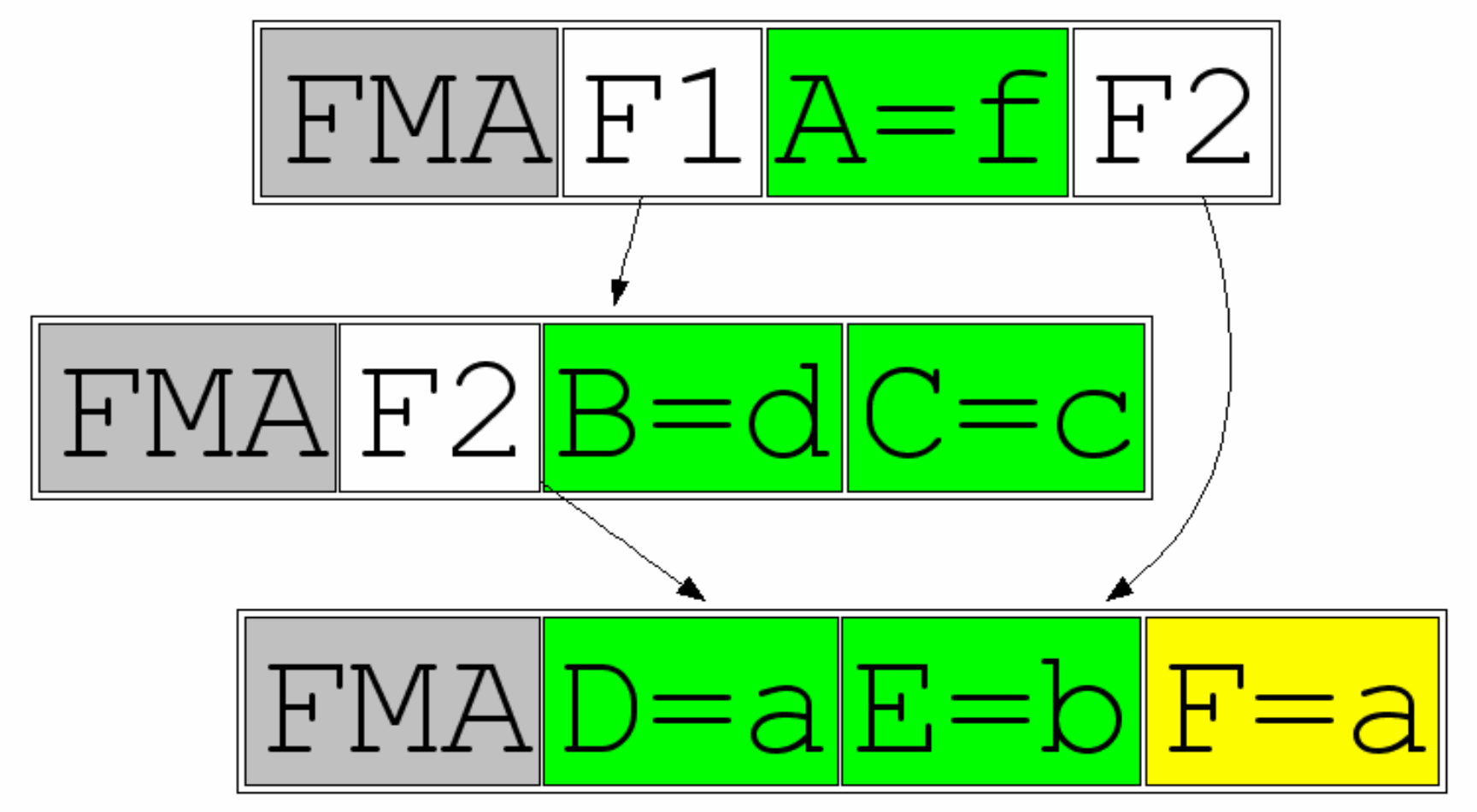
Formal: +ABDE+ABF+AC+DE+F
Actual: +abdf+cdf+ab-ef+c



Formal: +ABDE+ABF+AC+DE+F
Actual: +abdf+cdf+ab-ef+c

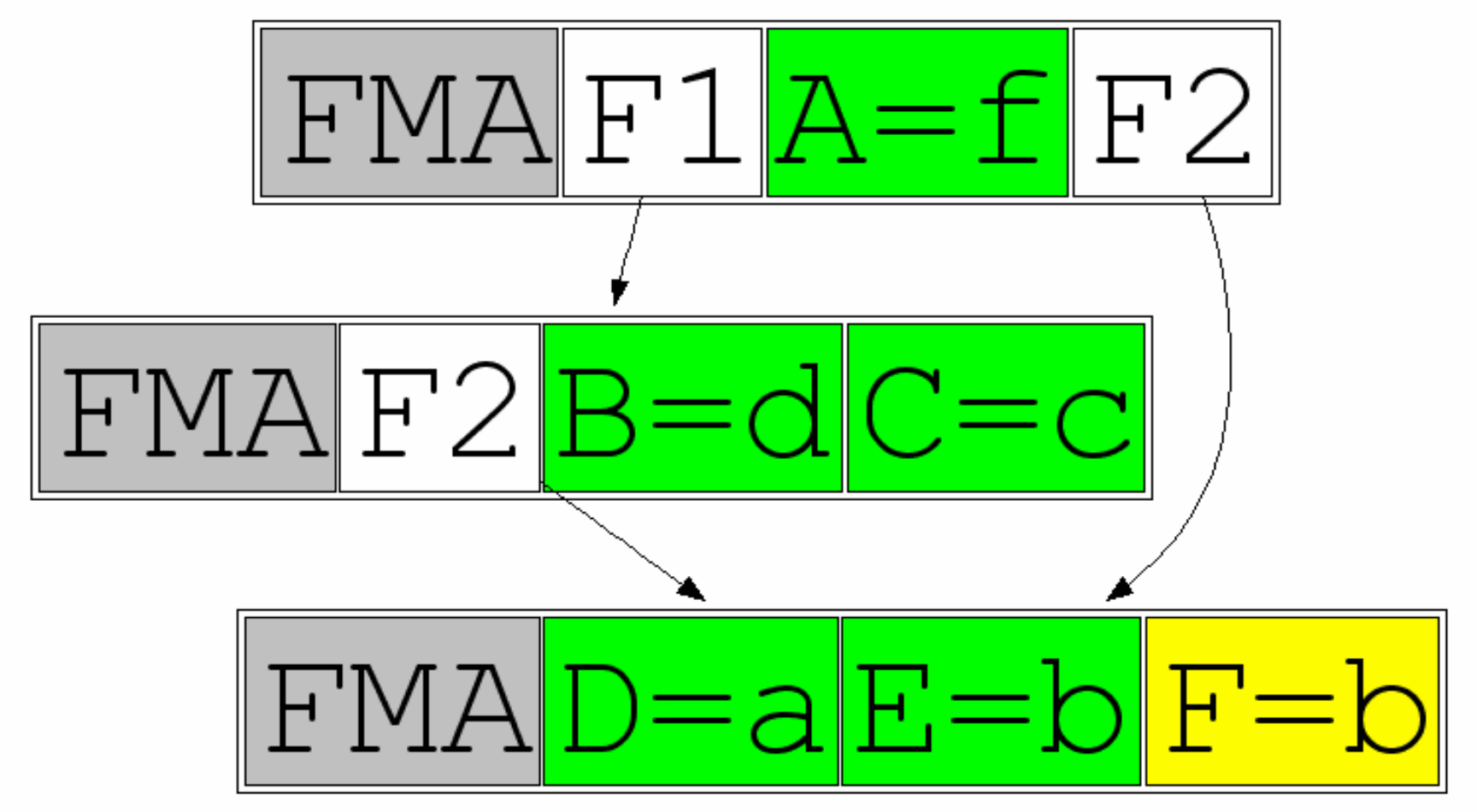


Formal: +ABDE+ABF+AC+DE+F
Actual: +abdf+cdf+ab-ef+c



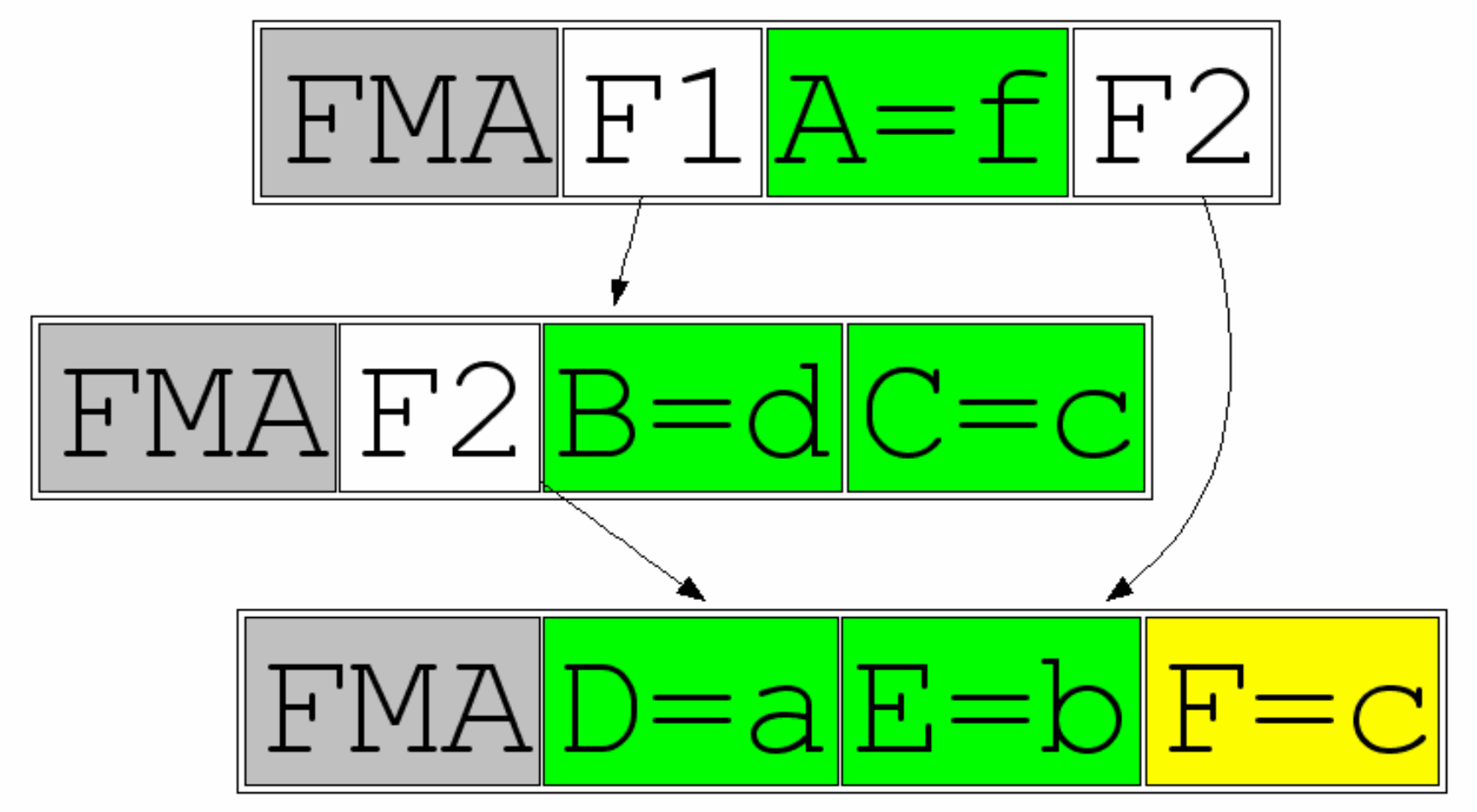
Formal: +ABDE+ABF+AC+DE+F

Actual: +abdf+cdf+ab-ef+c

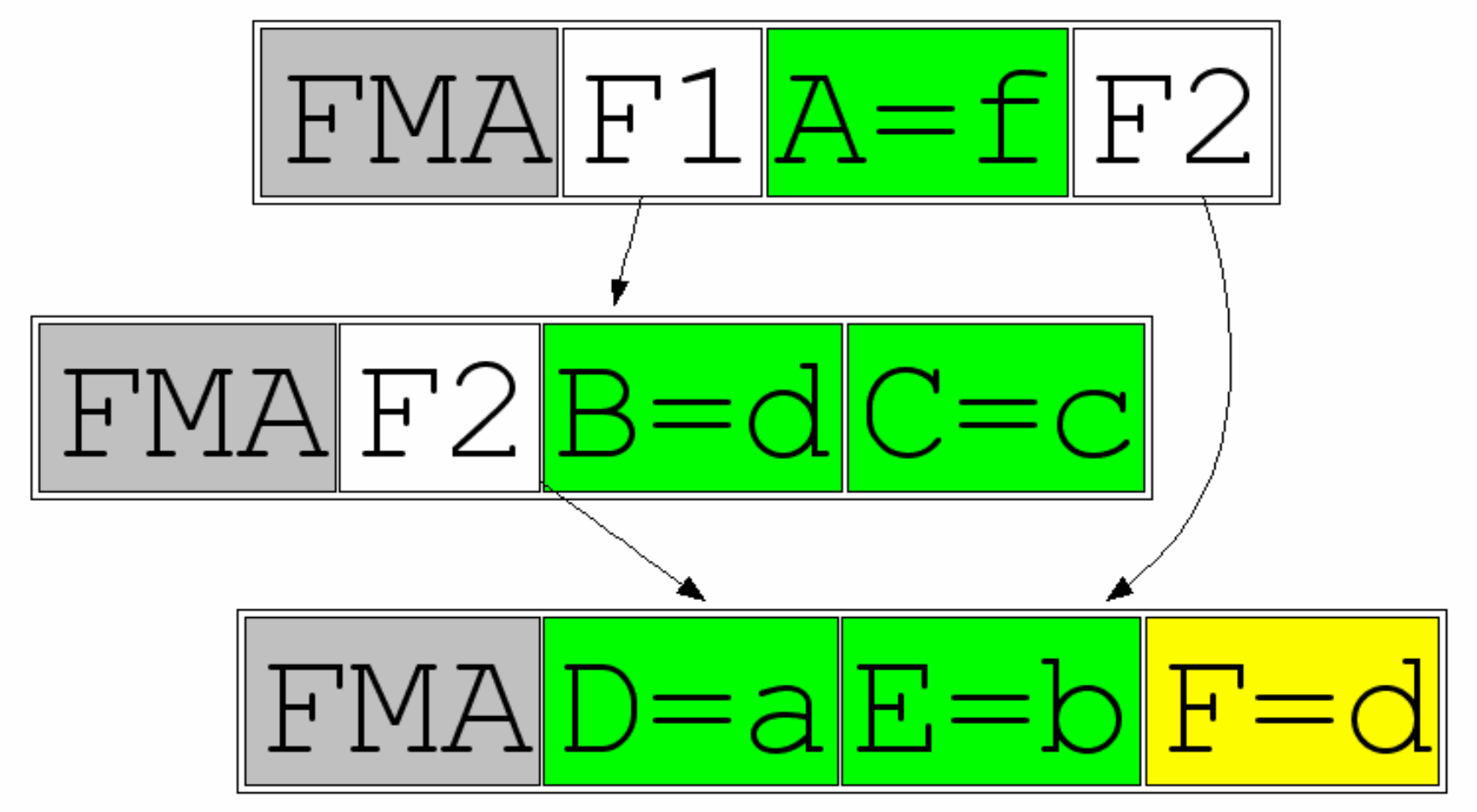


Formal: +ABDE+ABF+AC+DE+F

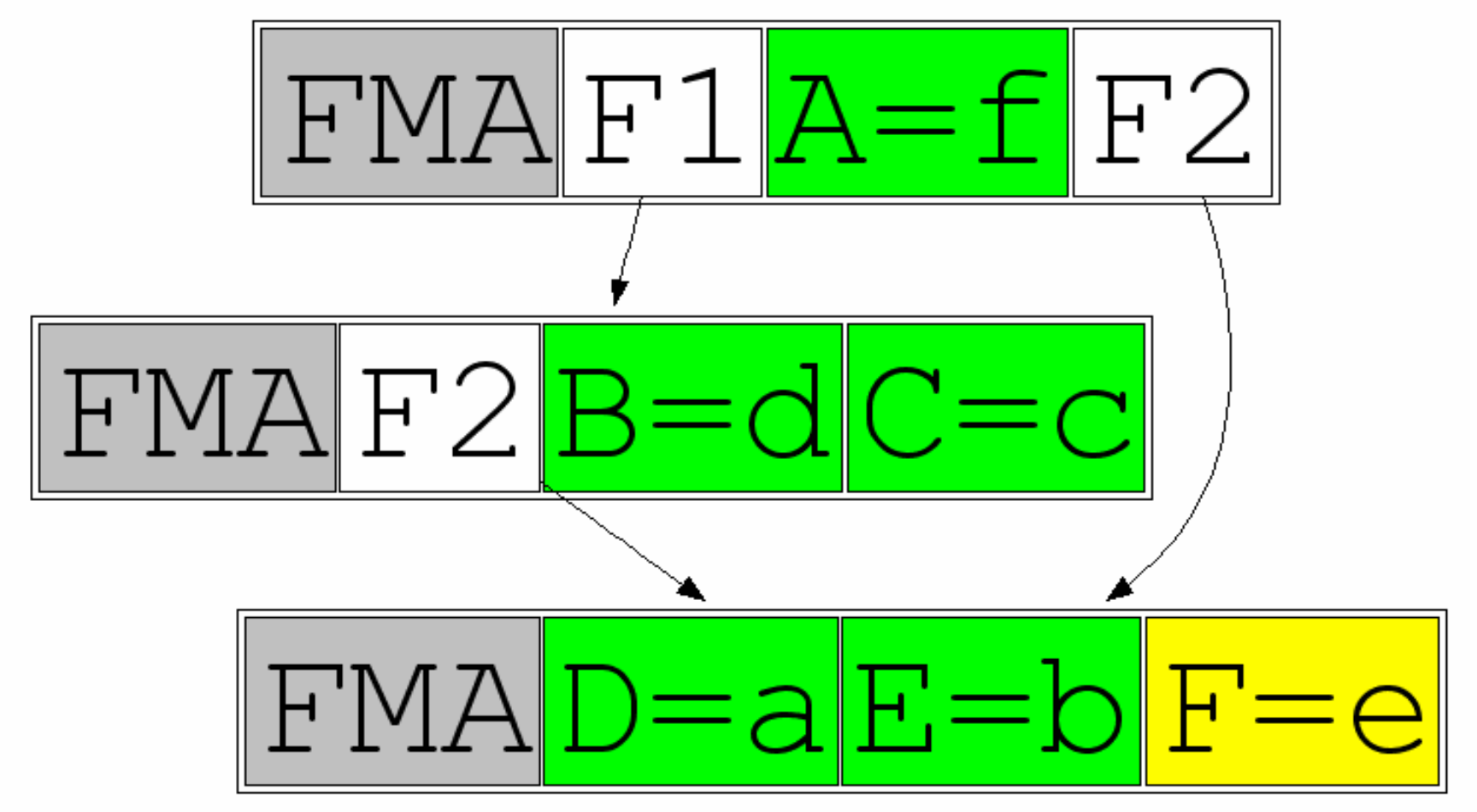
Actual: +abdf+cdf+ab-ef+c



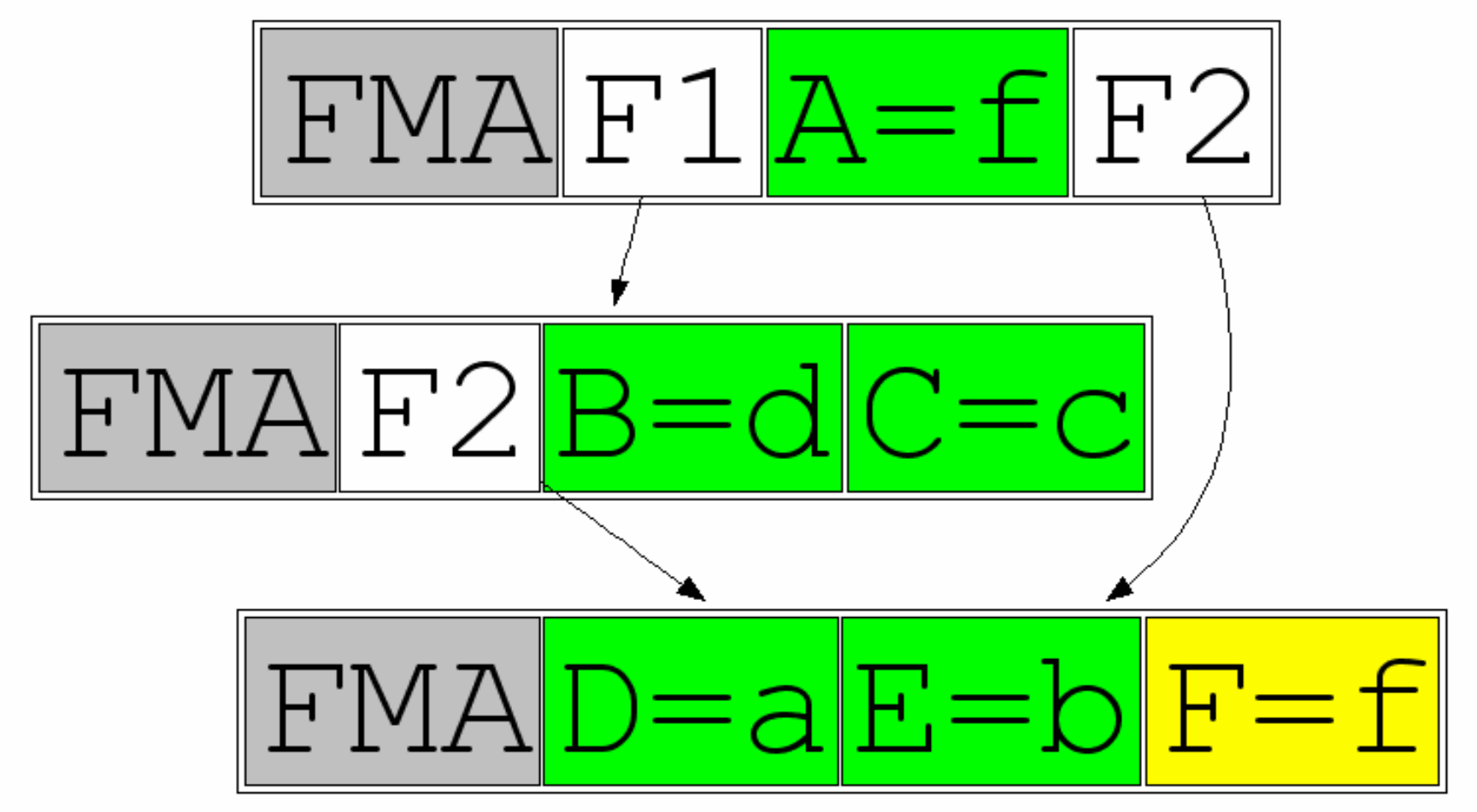
Formal: +ABDE+ABF+AC+DE+F
Actual: +abdf+cdf+ab-ef+c



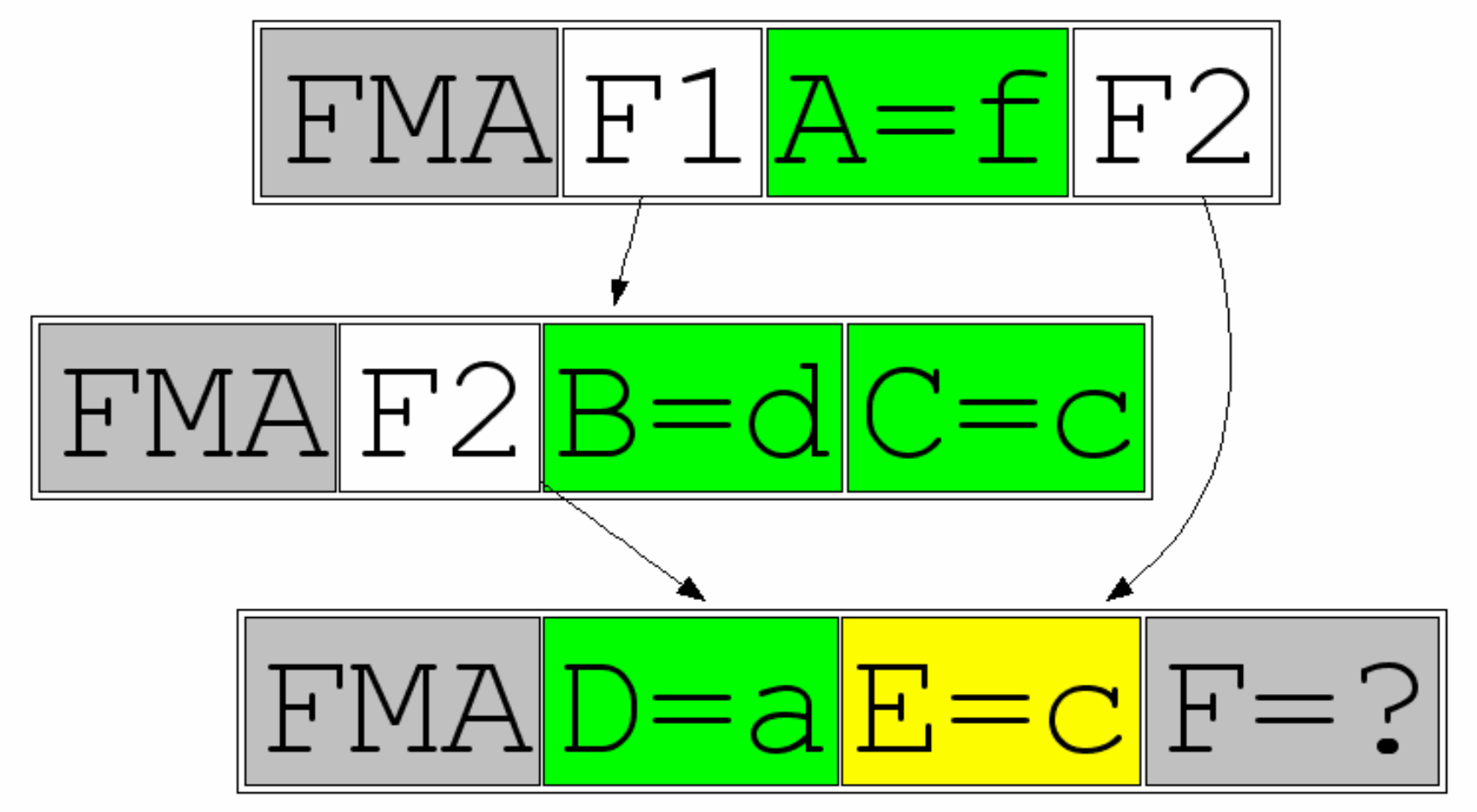
```
Formal: +ABDE+ABF+AC+DE+F
Actual: +abdf+cdf+ab-ef+c
```



Formal: +ABDE+ABF+AC+DE+F
Actual: +abdf+cdf+ab-ef+c

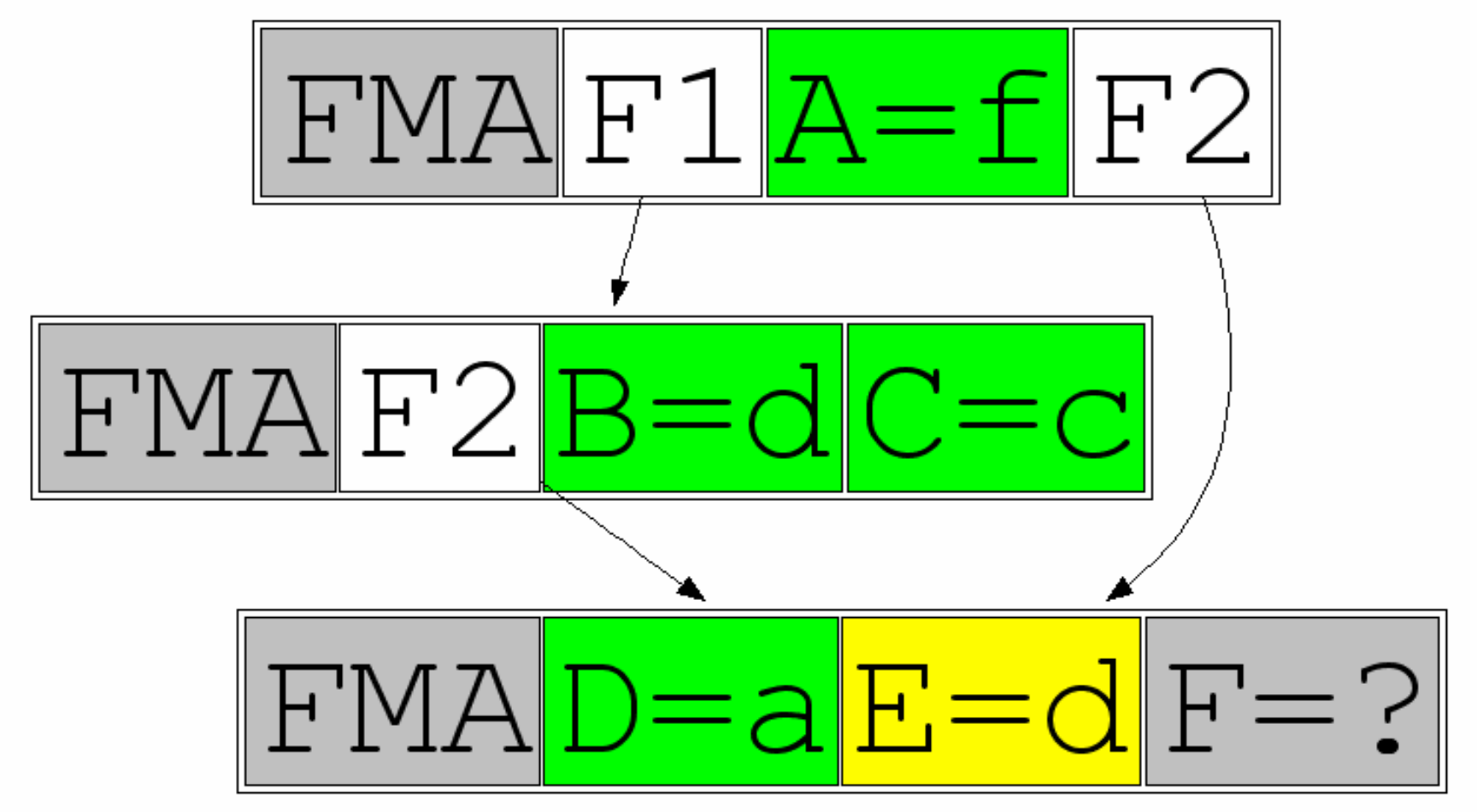


Formal: +ABDE+ABF+AC+DE+F
Actual: +abdf+cdf+ab-ef+c

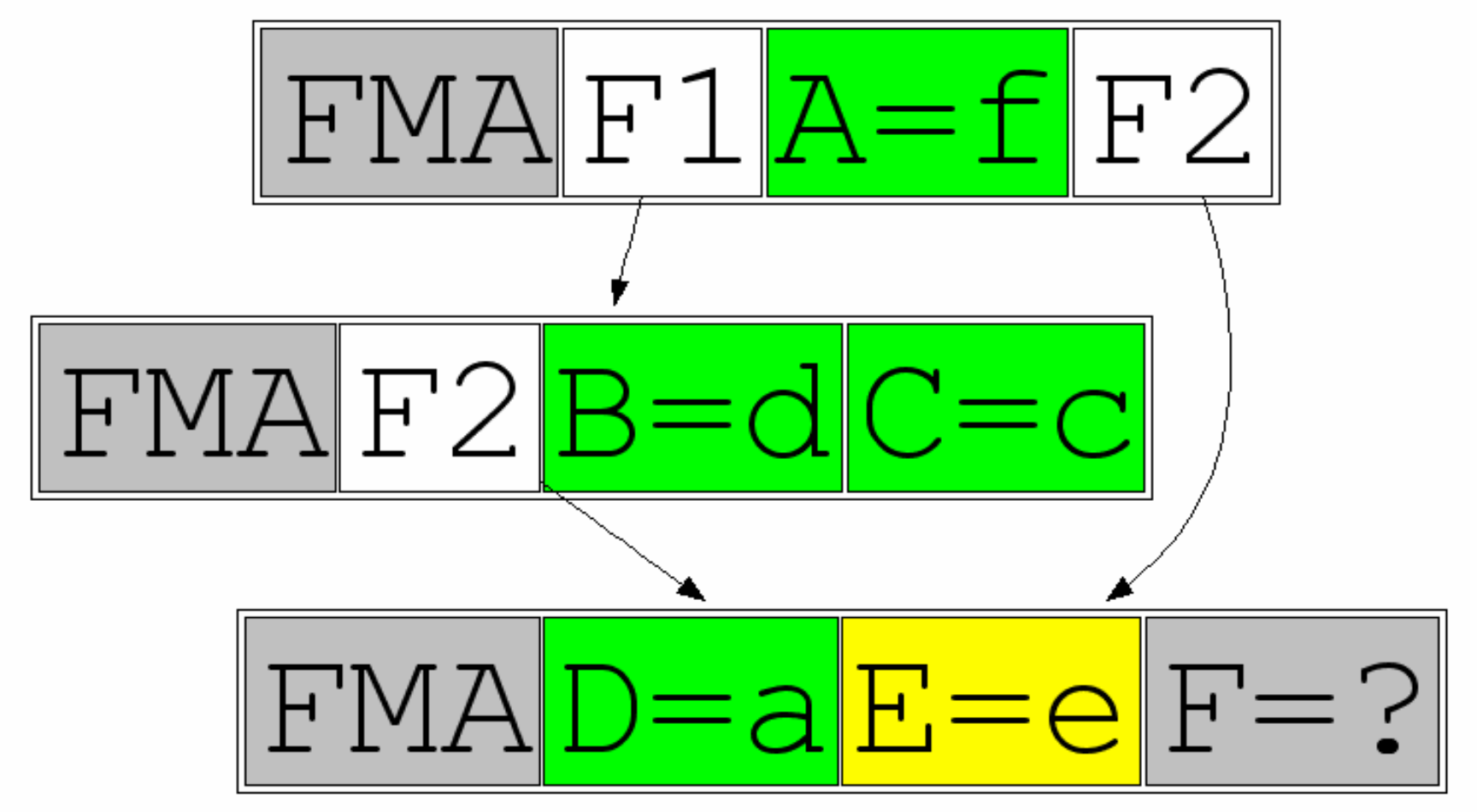


Formal: +ABDE+ABF+AC+DE+F

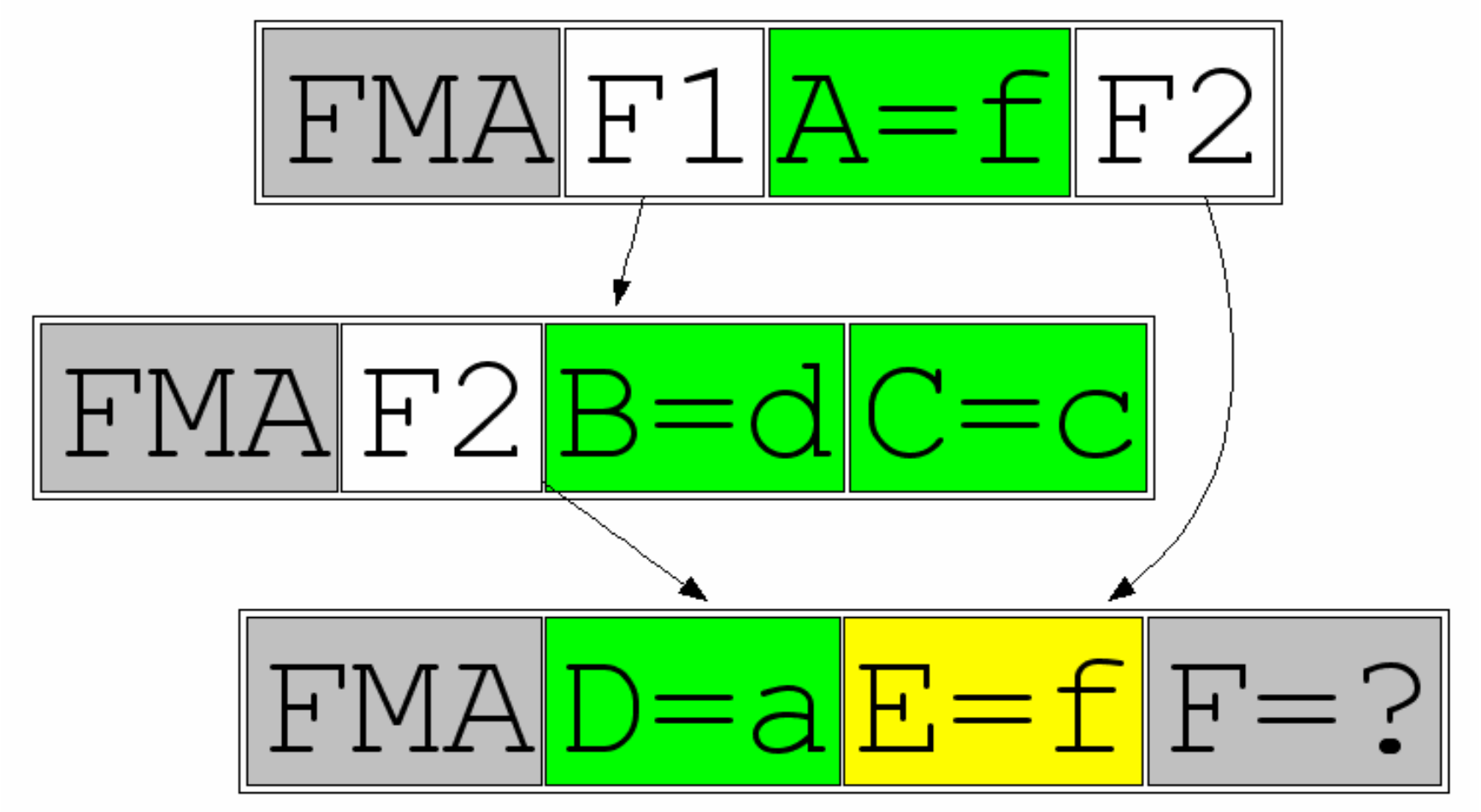
Actual: +abdf+cdf+ab-ef+c



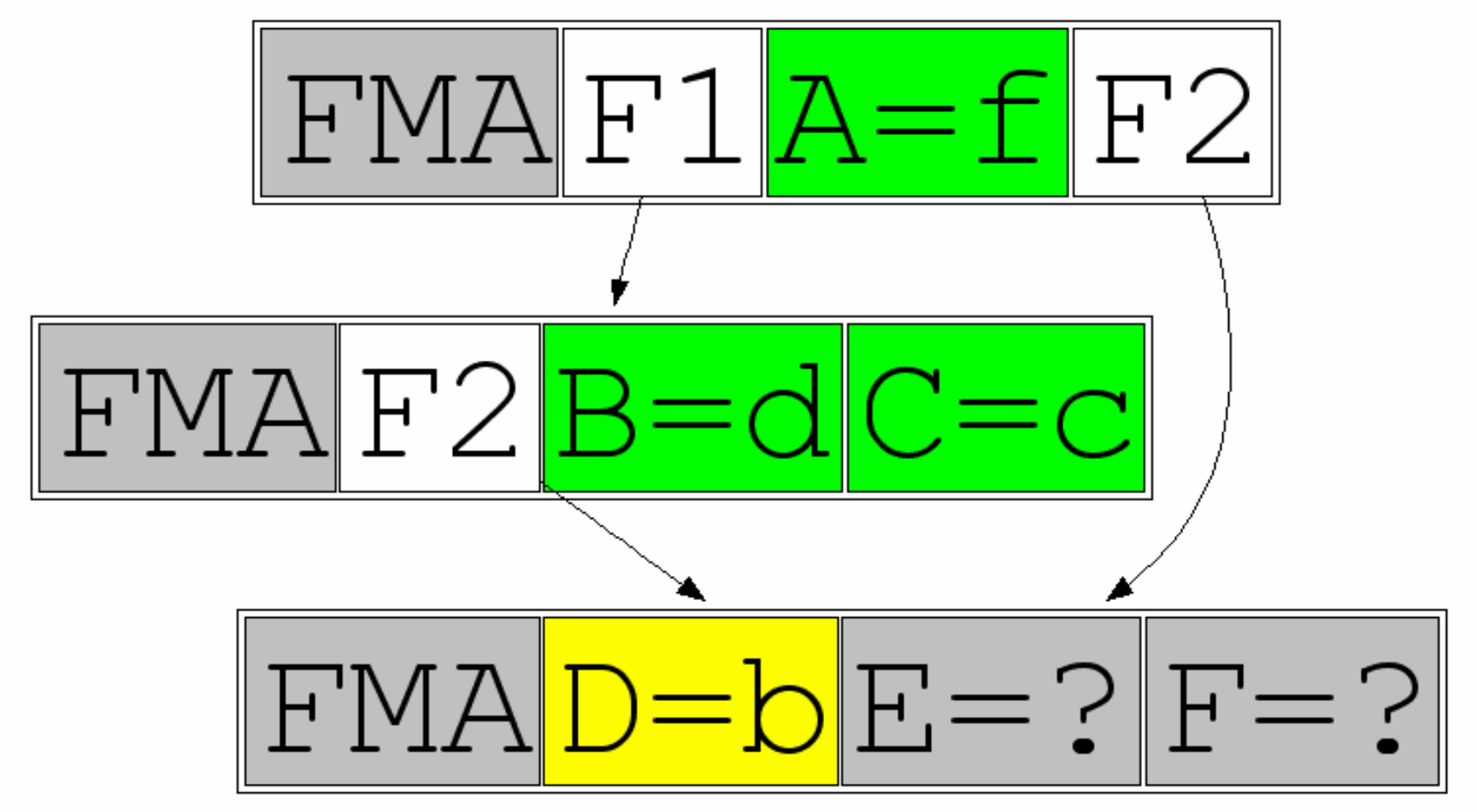
Formal: +ABDE+ABF+AC+DE+F
Actual: +abdf+cdf+ab-ef+c



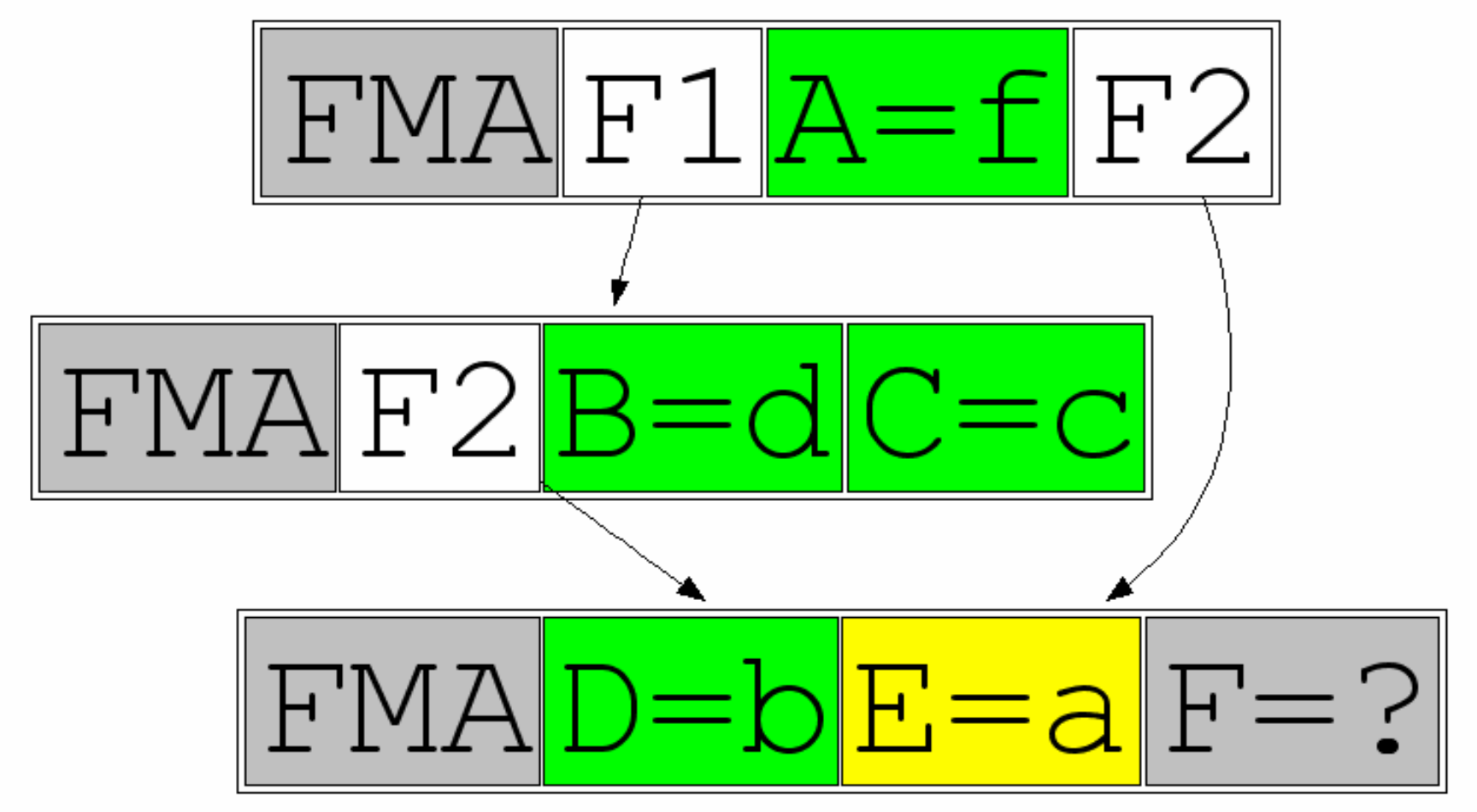
Formal: +ABDE+ABF+AC+DE+F
Actual: +abdf+cdf+ab-ef+c



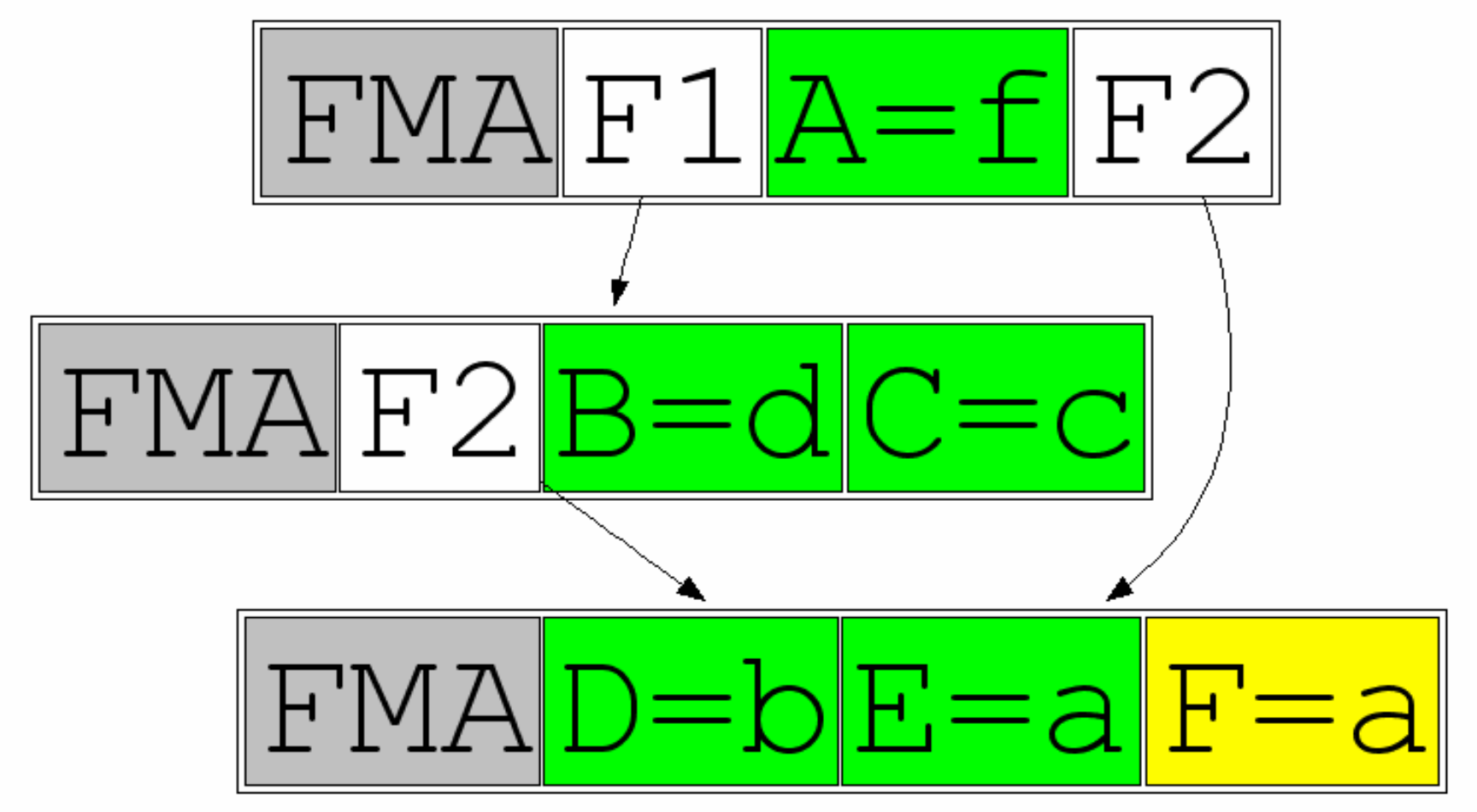
Formal: +ABDE+ABF+AC+DE+F
Actual: +abdf+cdf+ab-ef+c



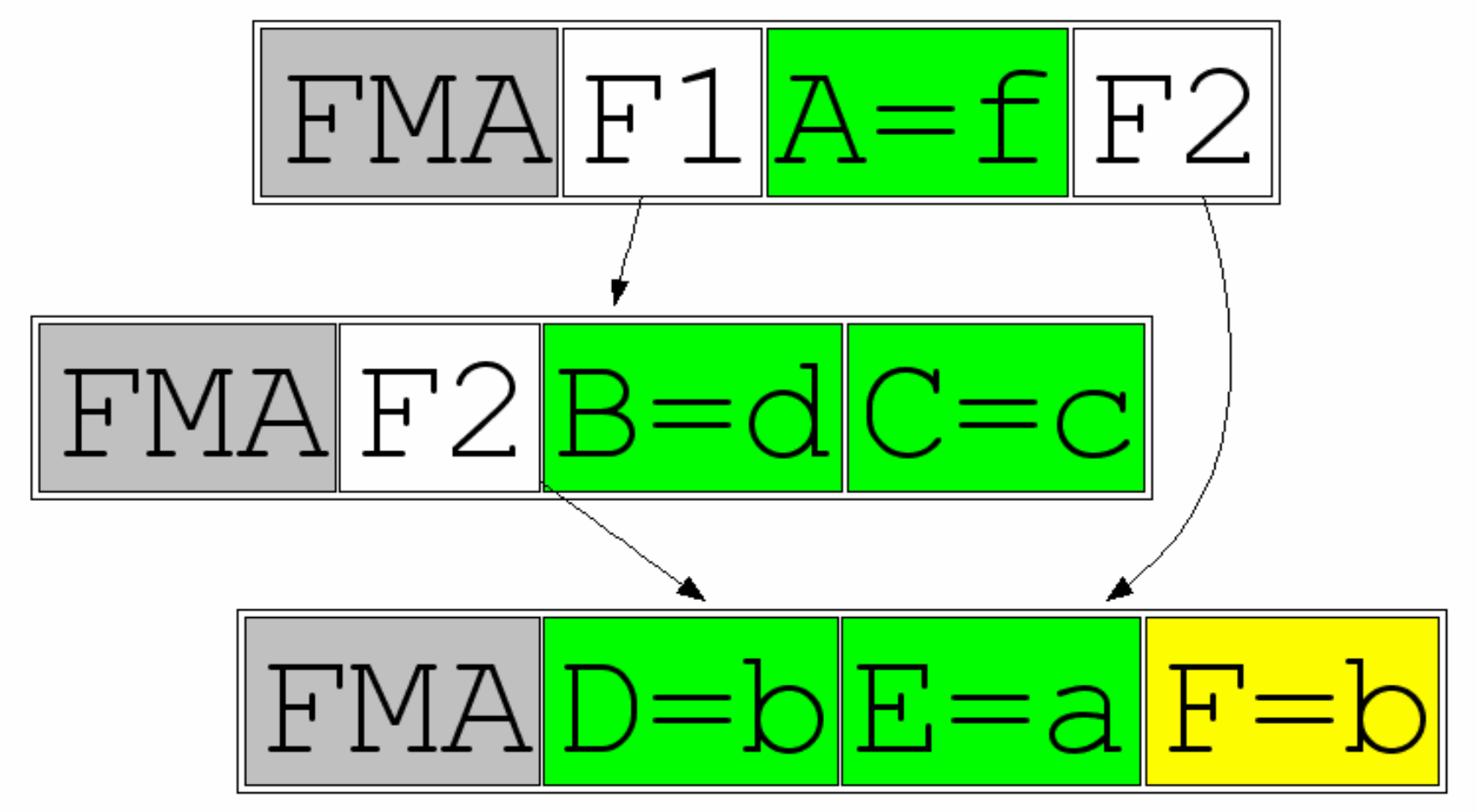
Formal: +ABDE+ABF+AC+DE+F
Actual: +abdf+cdf+ab-ef+c



Formal: +ABDE+ABF+AC+DE+F
Actual: +abdf+cdf+ab-ef+c

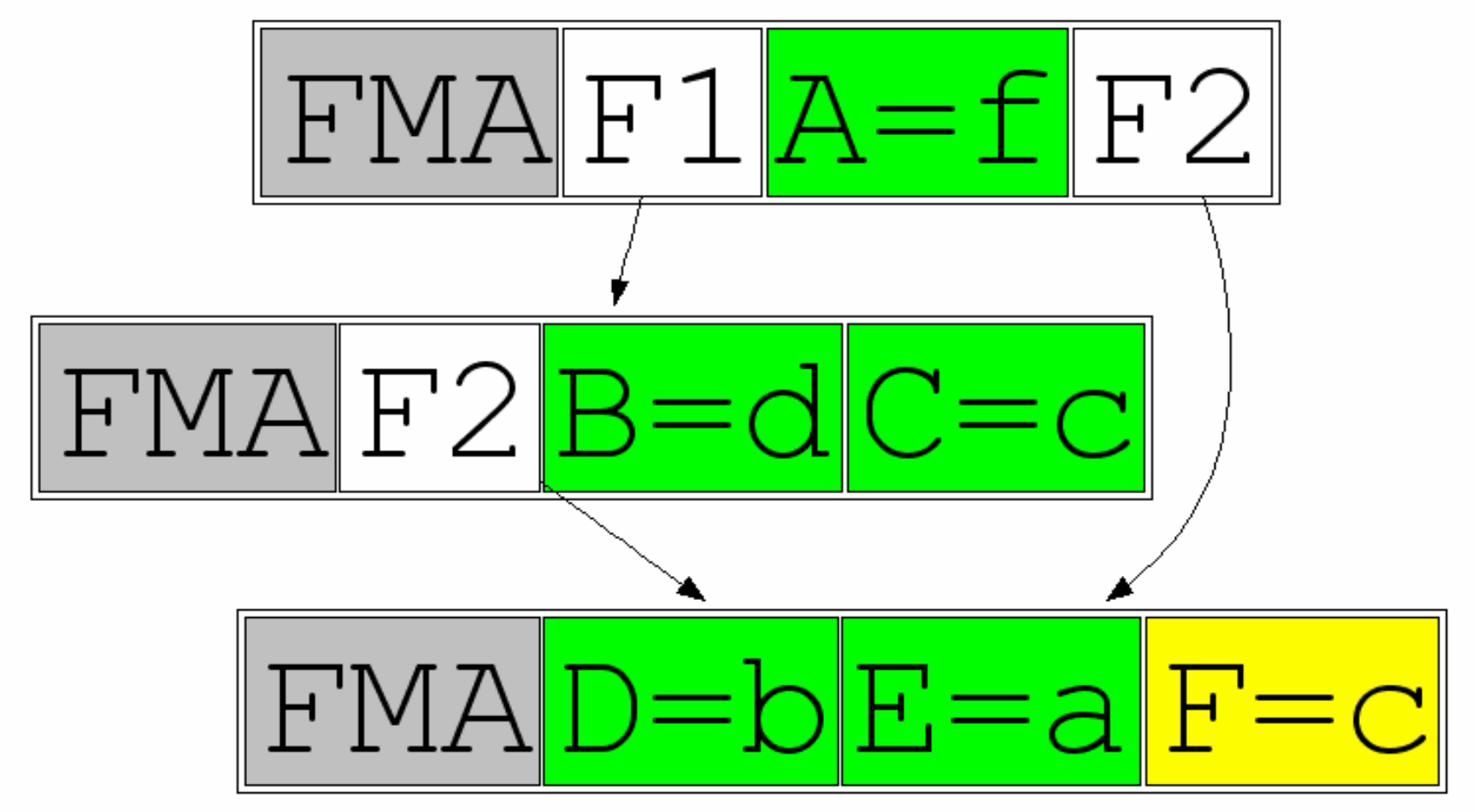


Formal: +ABDE+ABF+AC+DE+F
Actual: +abdf+cdf+ab-ef+c

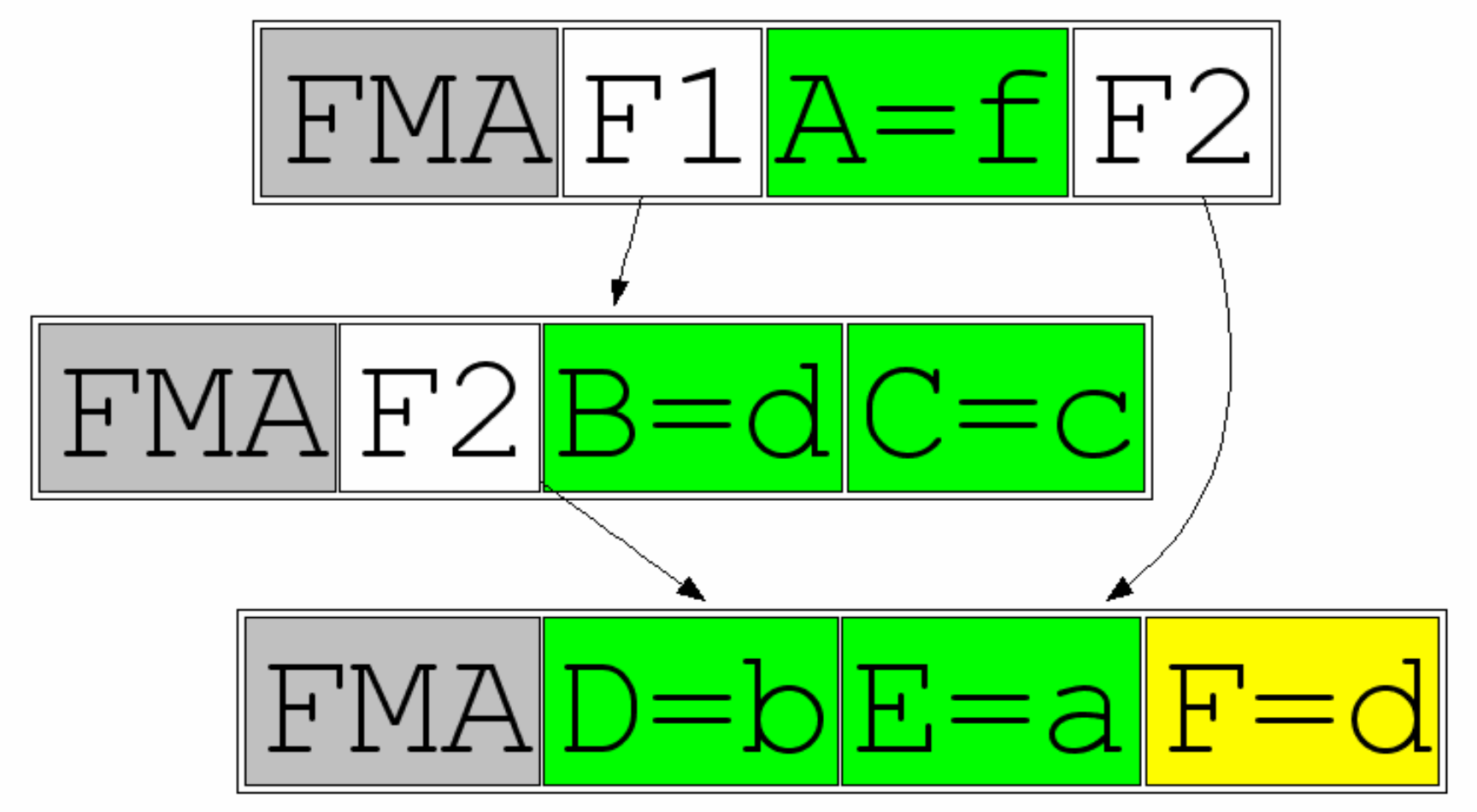


Formal: +ABDE+ABF+AC+DE+F

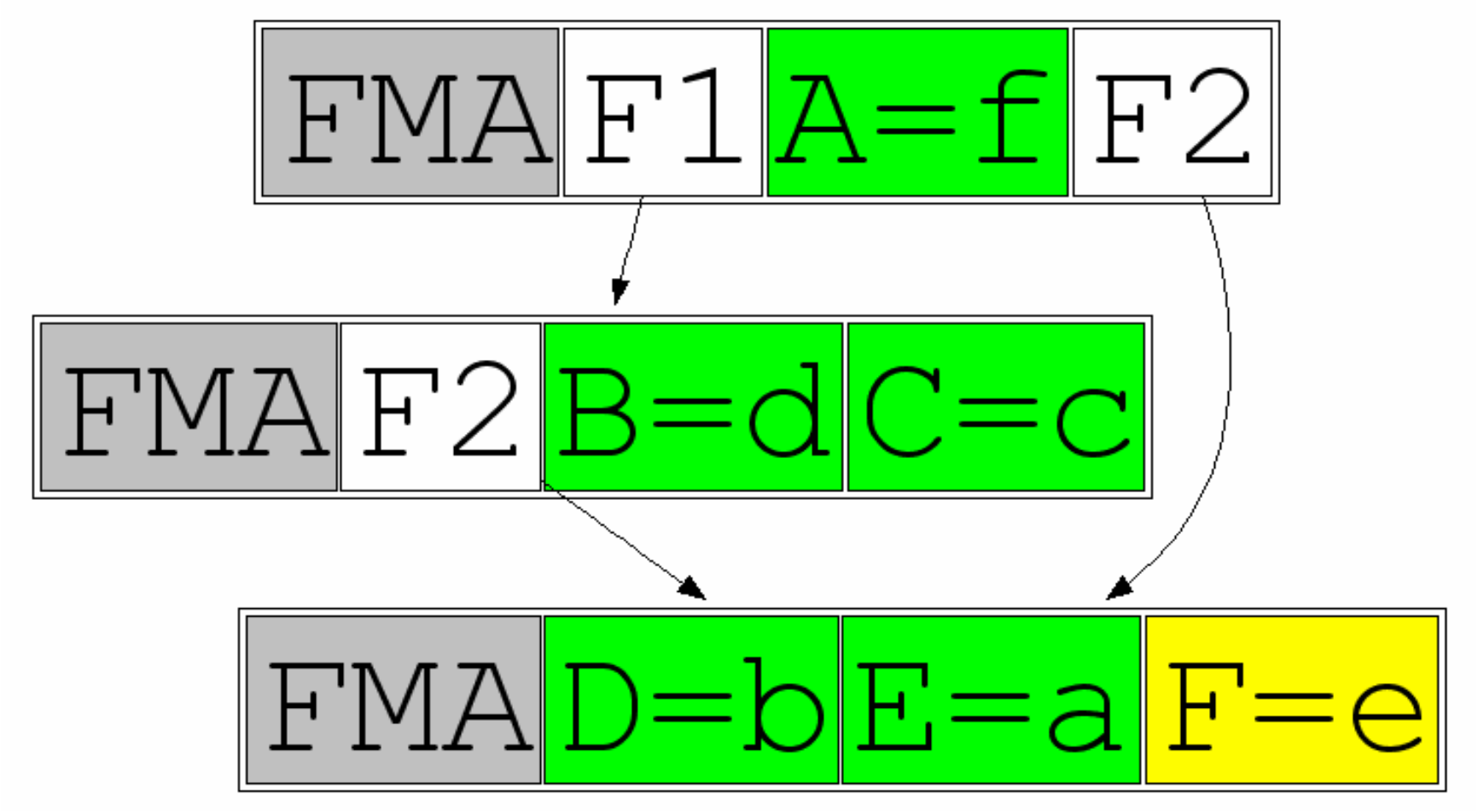
Actual: +abdf+cdf+ab-ef+c



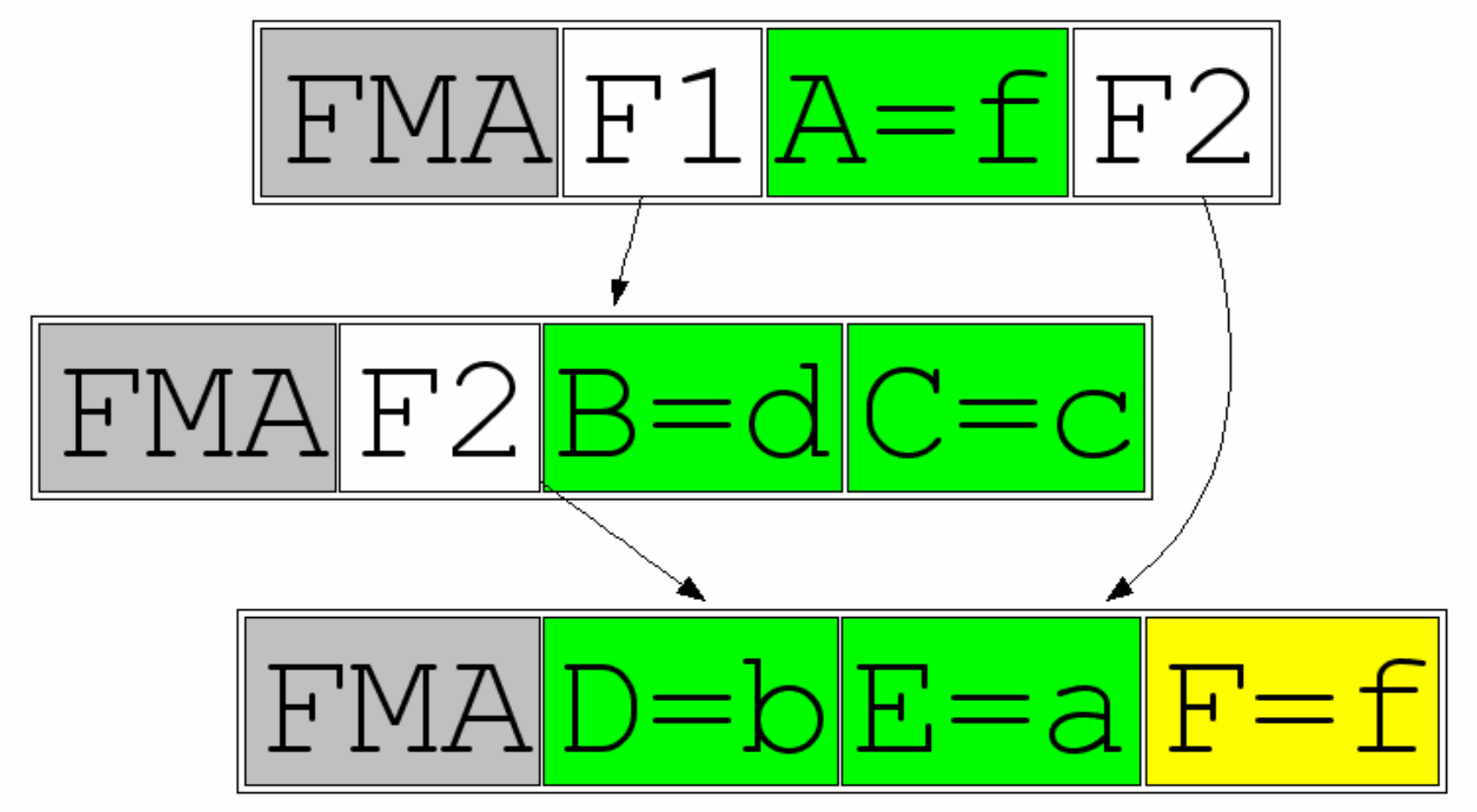
Formal: +ABDE+ABF+AC+DE+F
Actual: +abdf+cdf+ab-ef+c



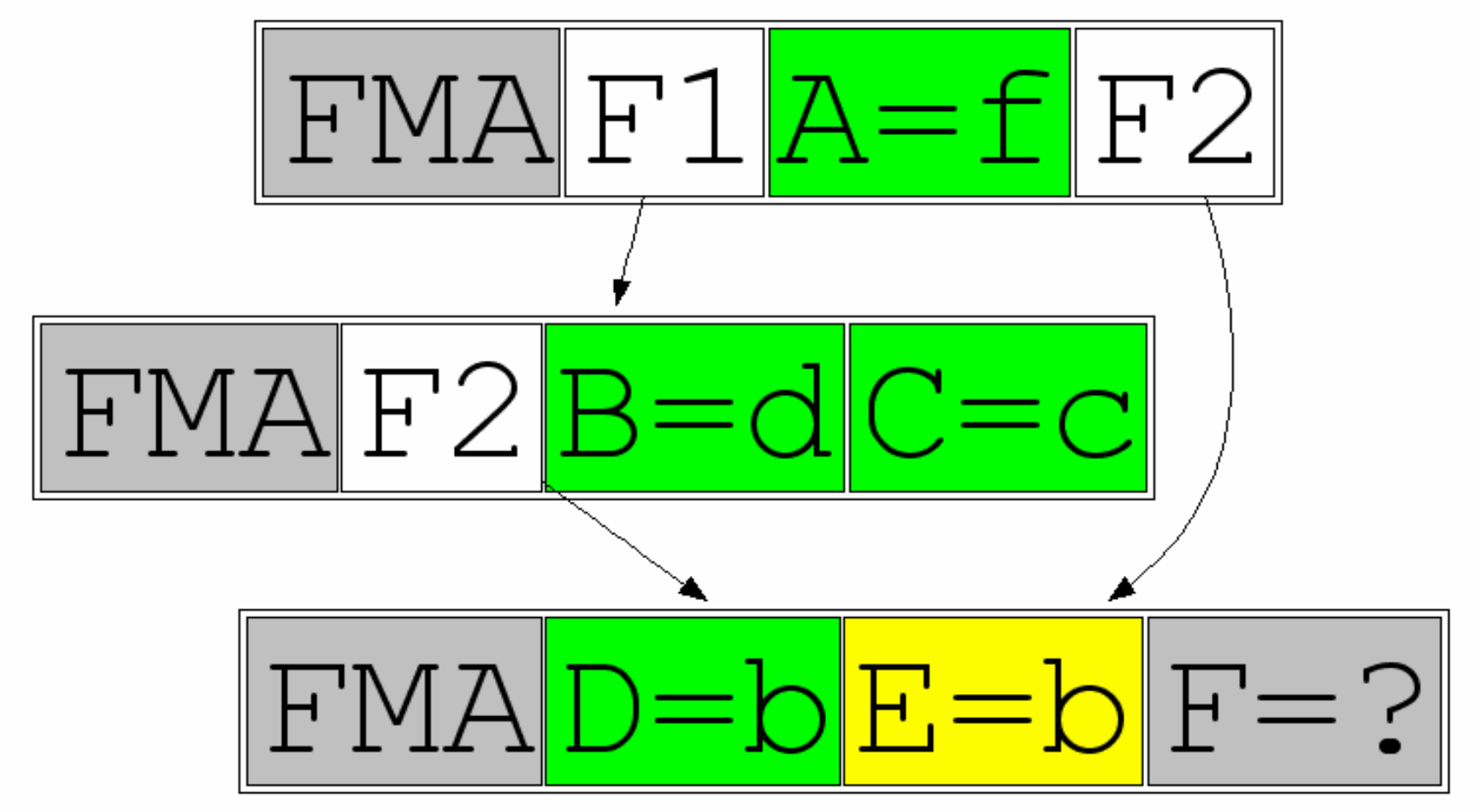
Formal: +ABDE+ABF+AC+DE+F
Actual: +abdf+cdf+ab-ef+c



Formal: +ABDE+ABF+AC+DE+F
Actual: +abdf+cdf+ab-ef+c

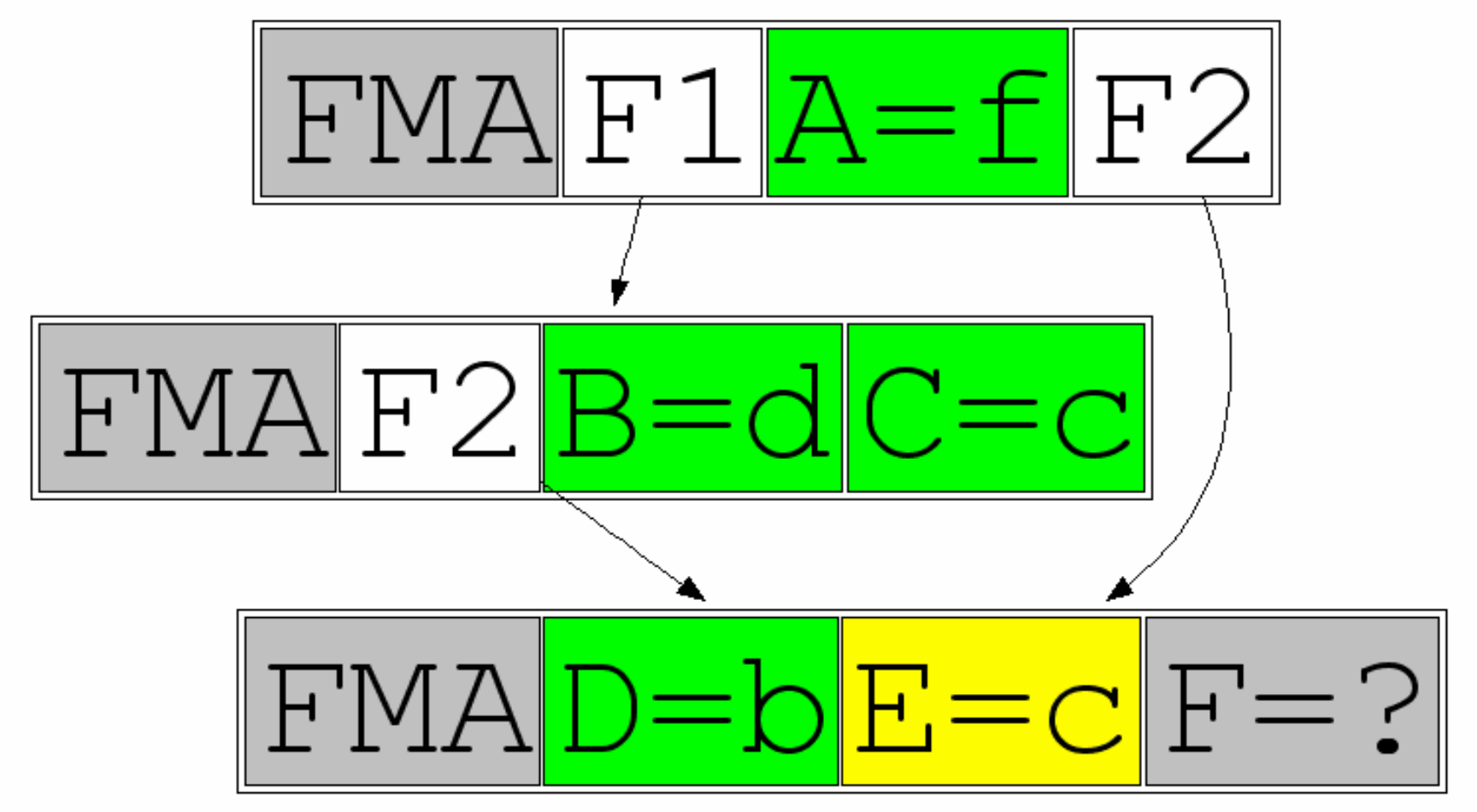


Formal: +ABDE+ABF+AC+DE+F
Actual: +abdf+cdf+ab-ef+c

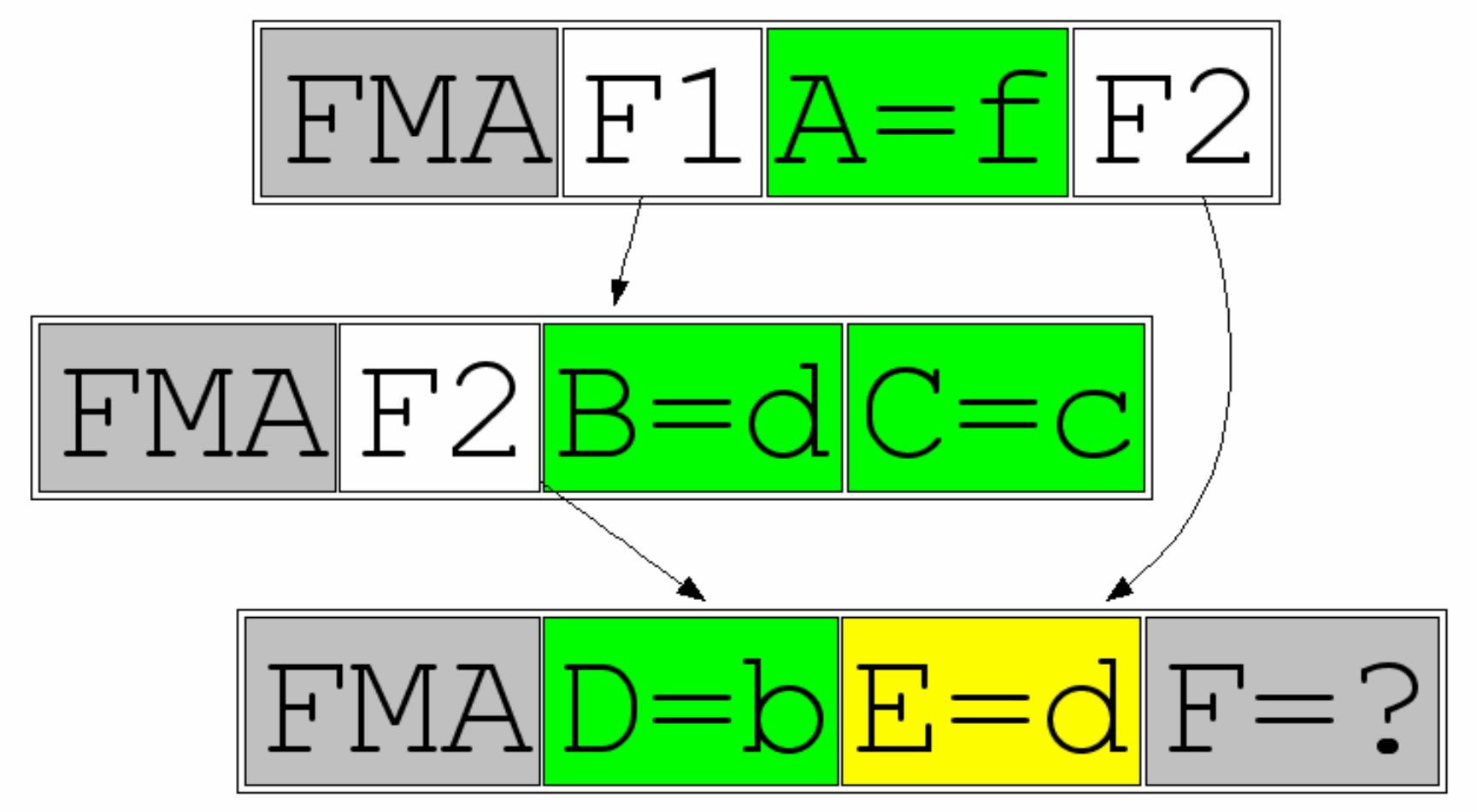


Formal: +ABDE+ABF+AC+DE+F

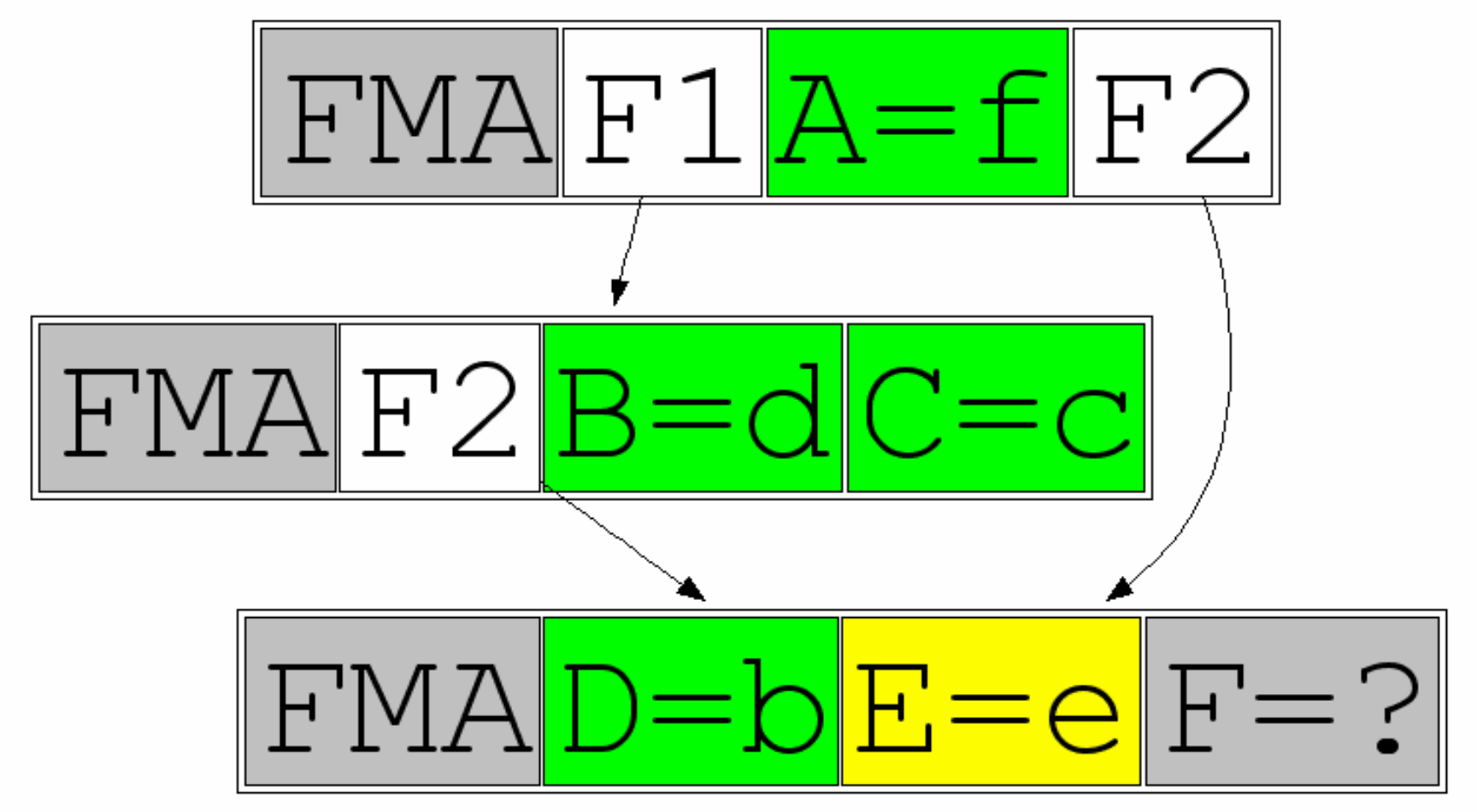
Actual: +abdf+cdf+ab-ef+c



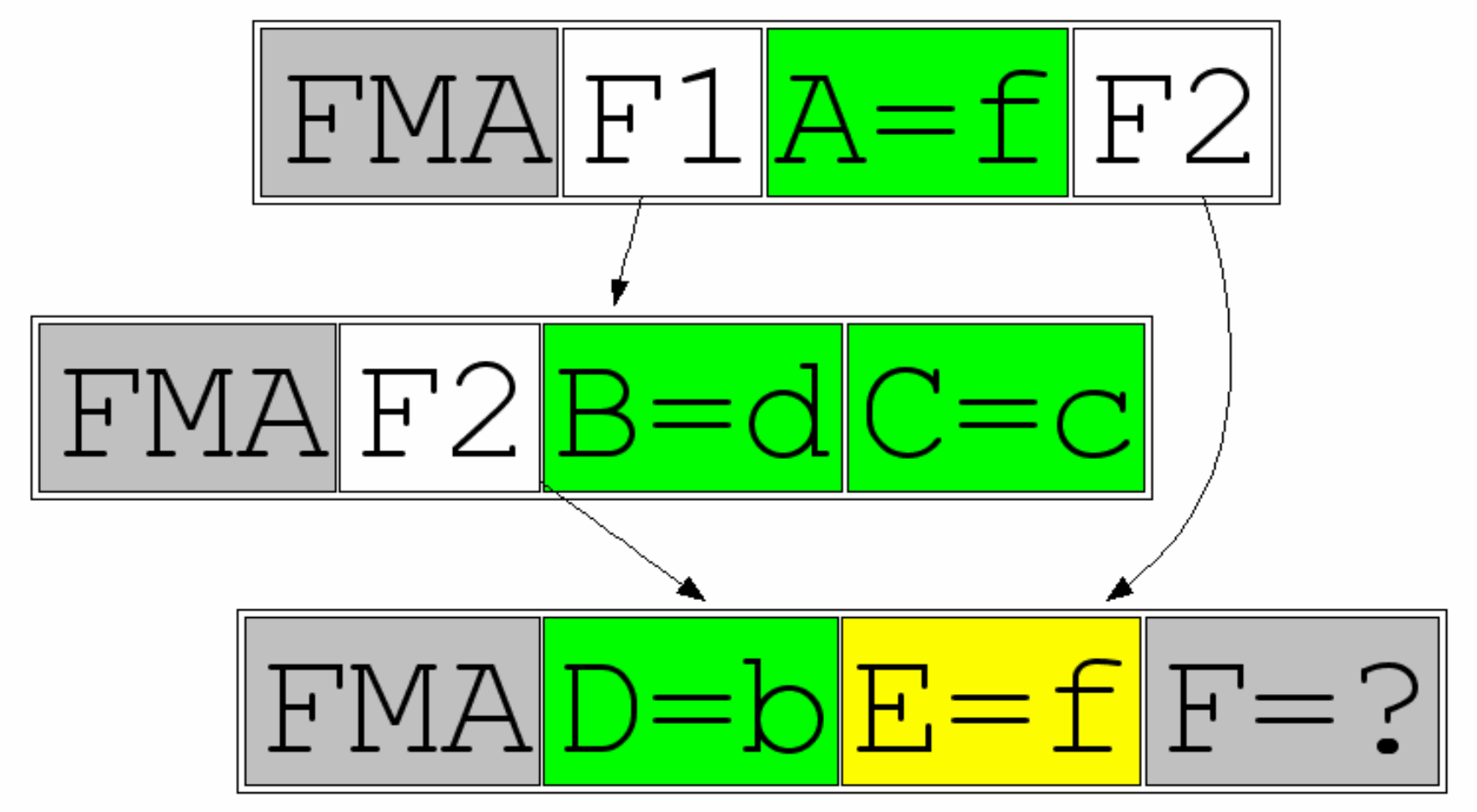
Formal: +ABDE+ABF+AC+DE+F
Actual: +abdf+cdf+ab-ef+c



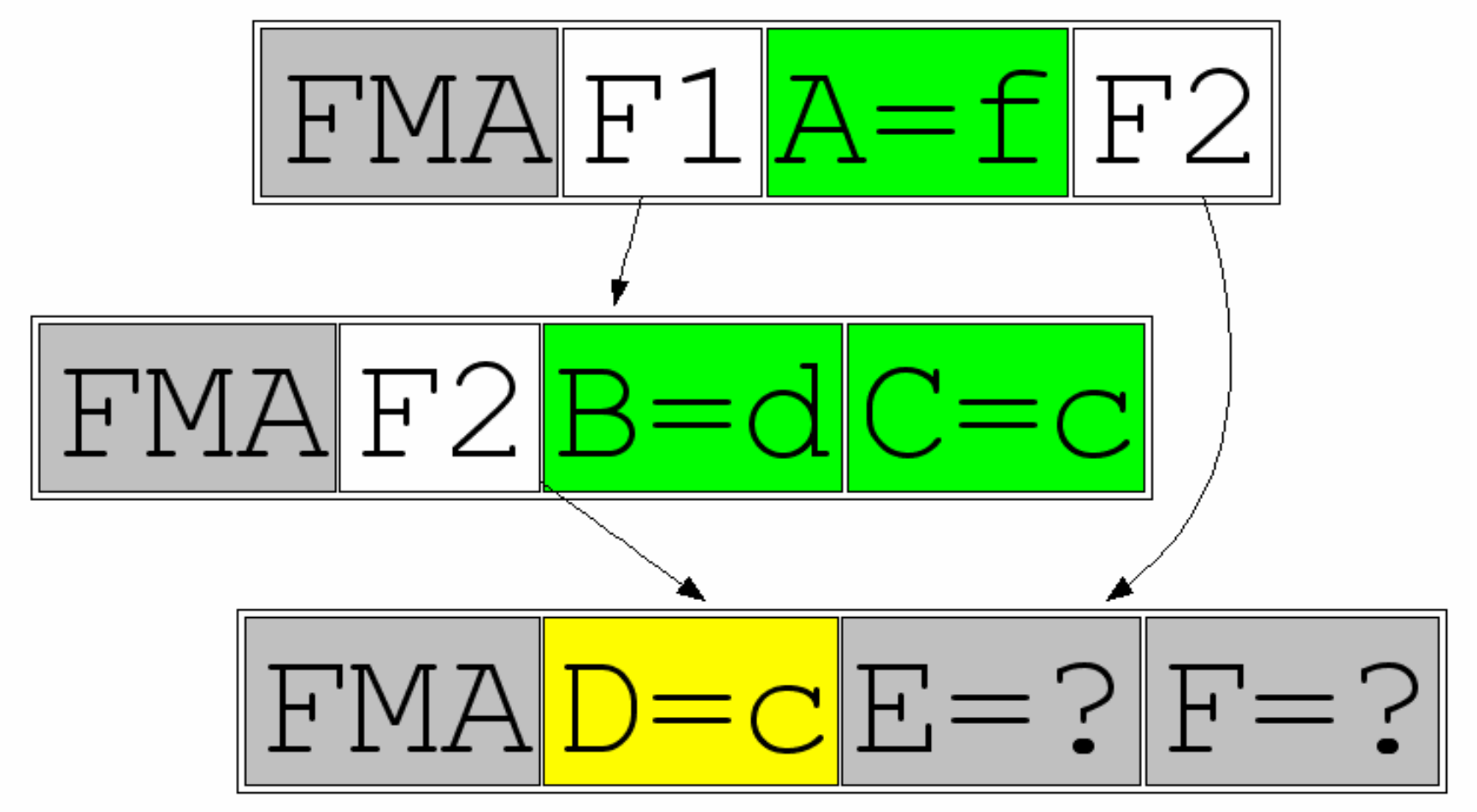
Formal: +ABDE+ABF+AC+DE+F
Actual: +abdf+cdf+ab-ef+c



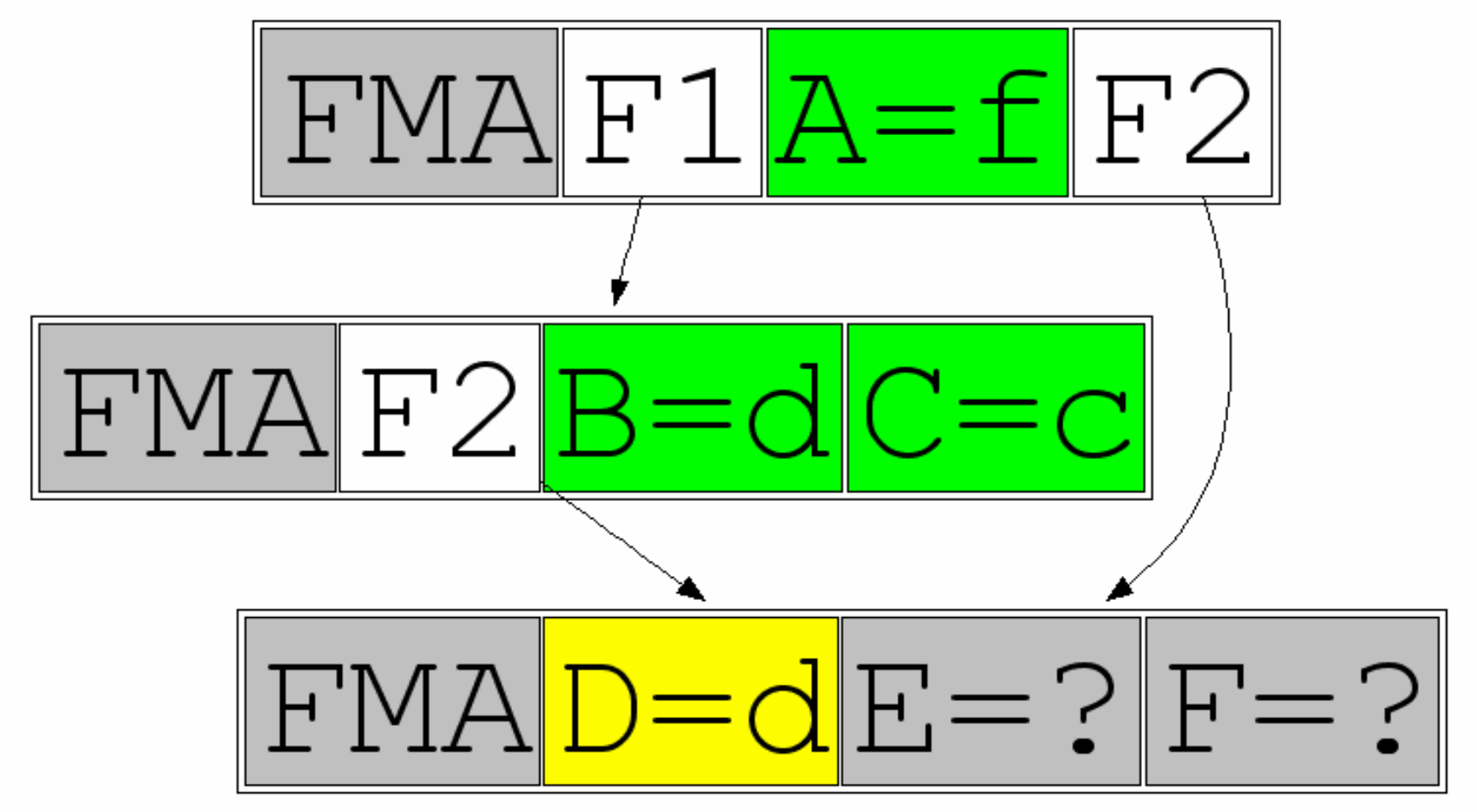
Formal: +ABDE+ABF+AC+DE+F
Actual: +abdf+cdf+ab-ef+c



Formal: +ABDE+ABF+AC+DE+F
Actual: +abdf+cdf+ab-ef+c

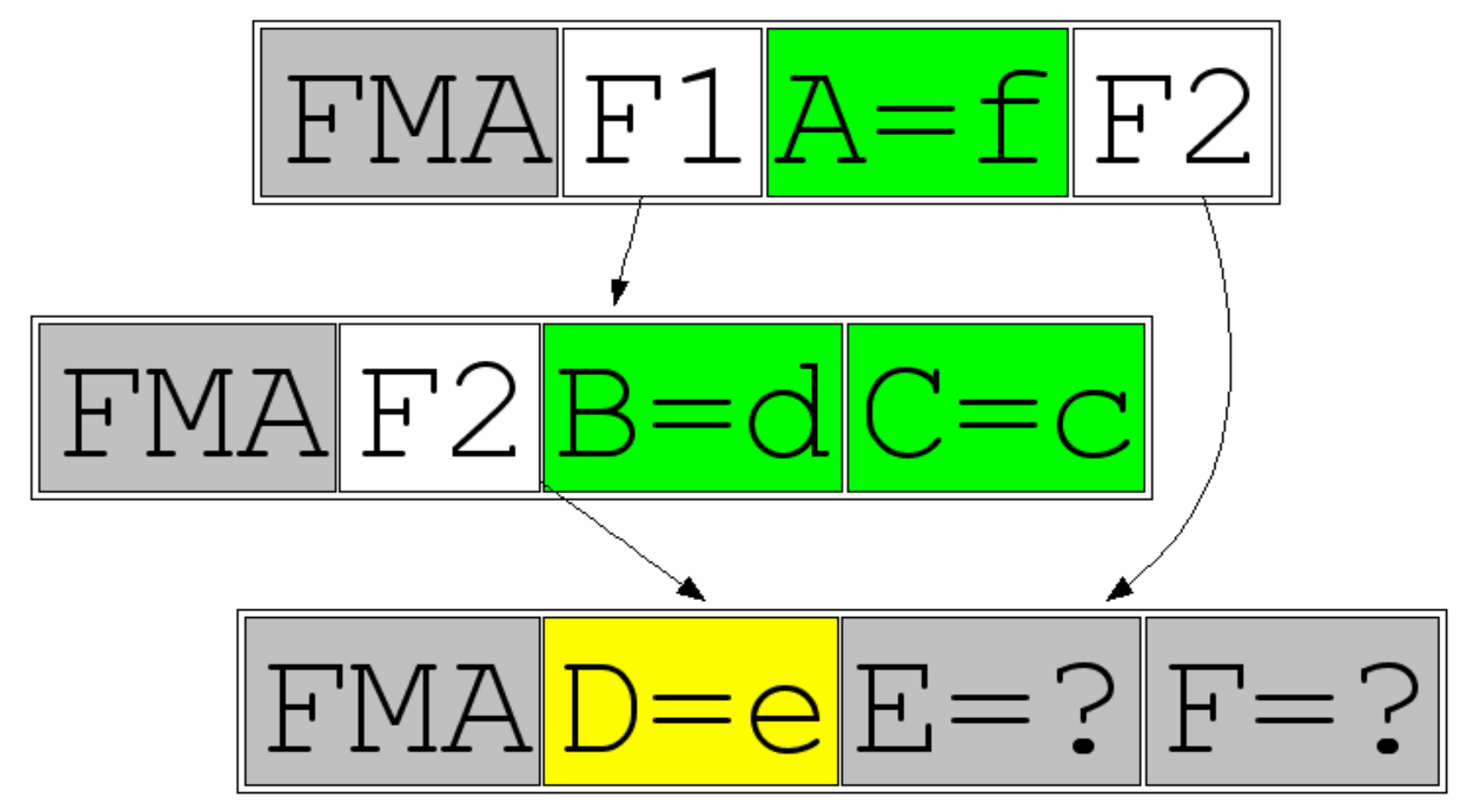


Formal: +ABDE+ABF+AC+DE+F
Actual: +abdf+cdf+ab-ef+c

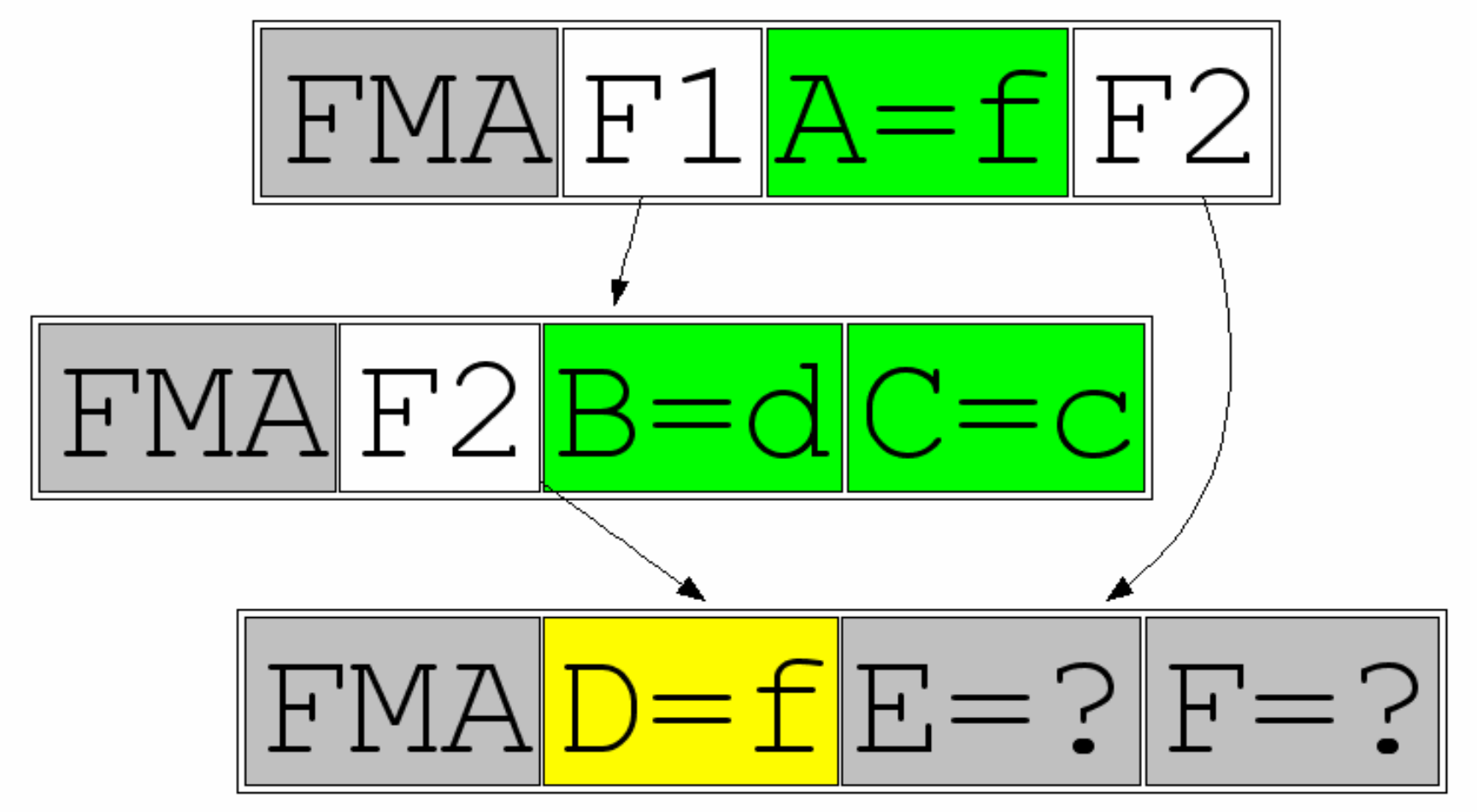


Formal: +ABDE+ABF+AC+DE+F

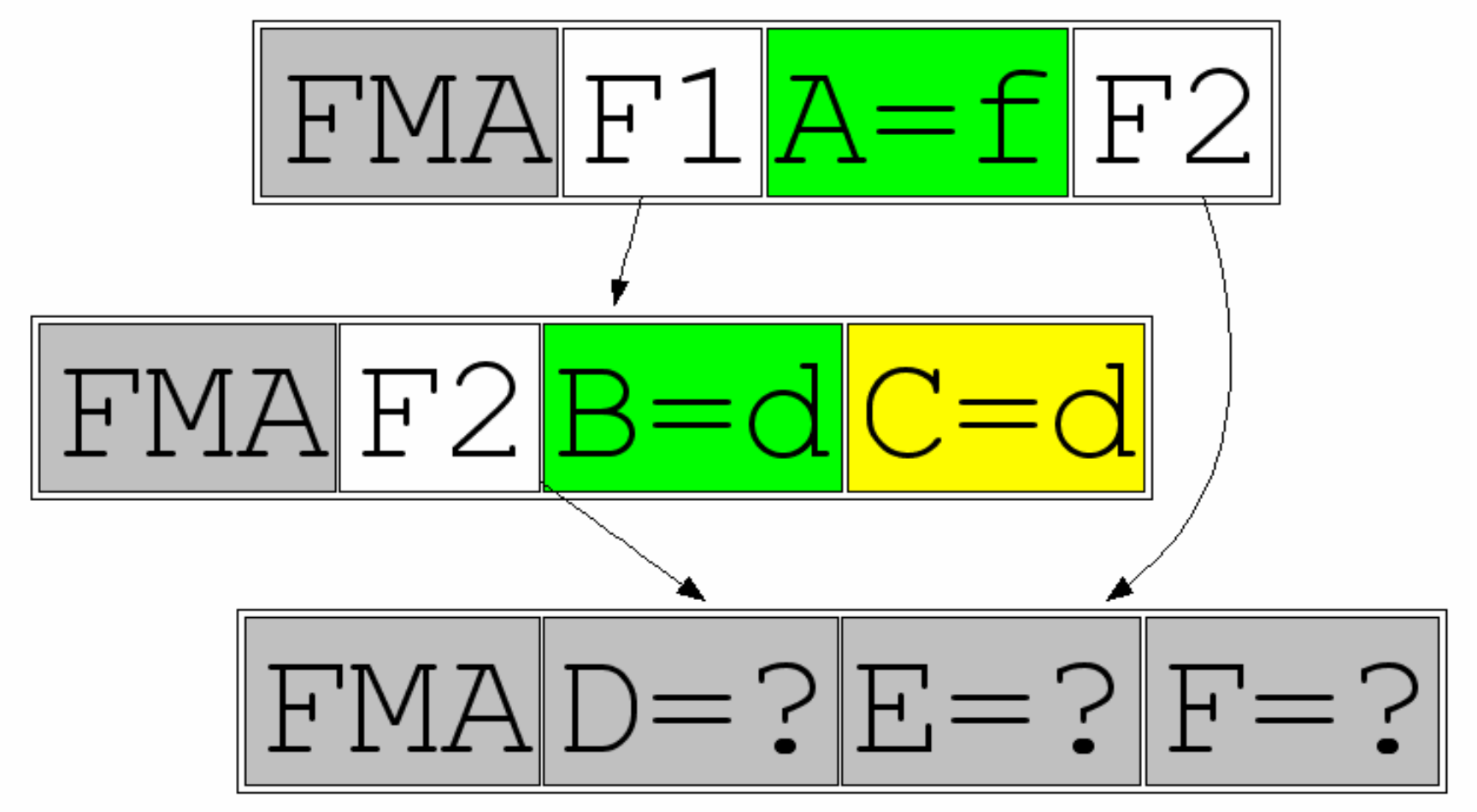
Actual: +abdf+cdf+ab-ef+c



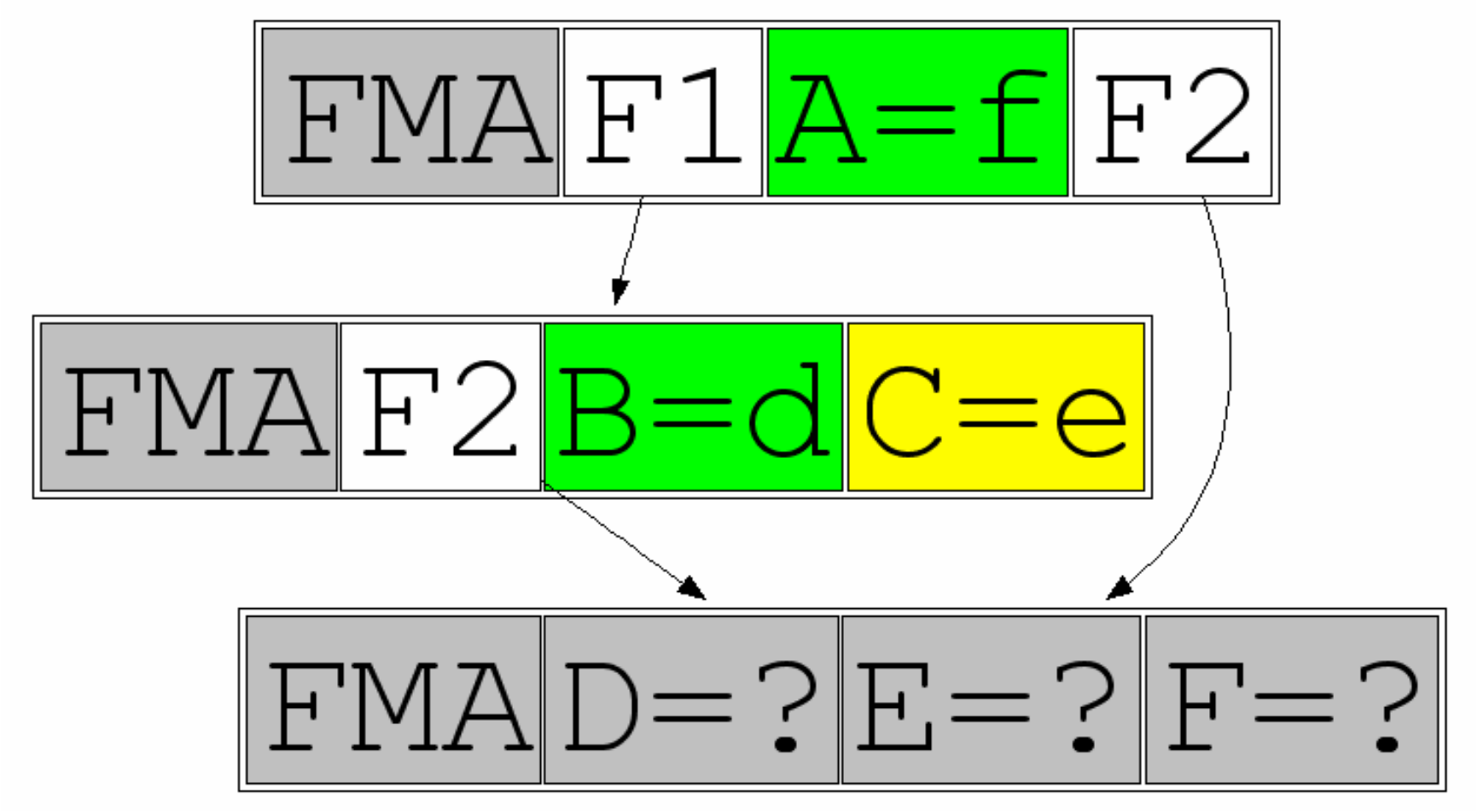
Formal: +ABDE+ABF+AC+DE+F
Actual: +abdf+cdf+ab-ef+c



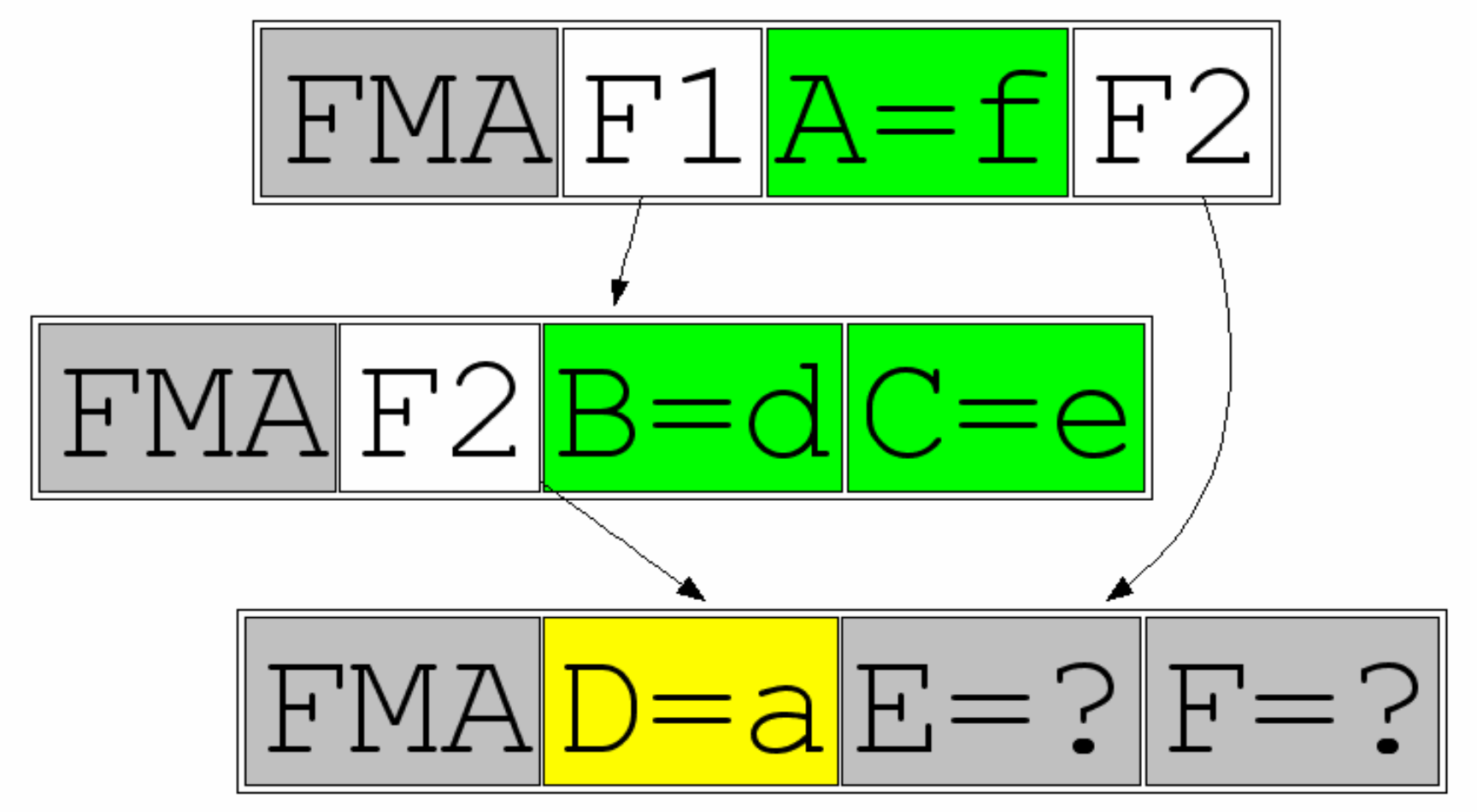
Formal: +ABDE+ABF+AC+DE+F
Actual: +abdf+cdf+ab-ef+c



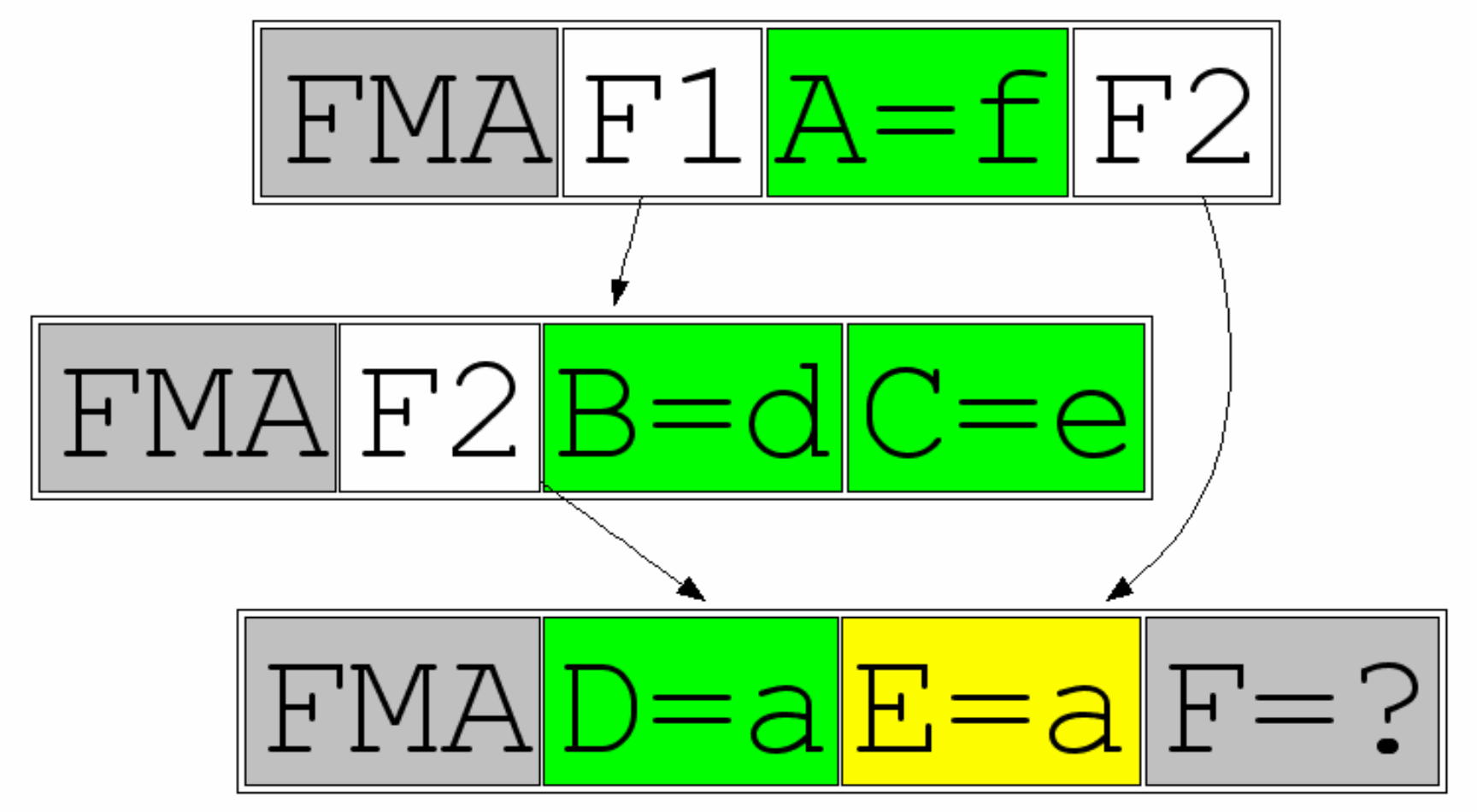
Formal: +ABDE+ABF+AC+DE+F
Actual: +abdf+cdf+ab-ef+c



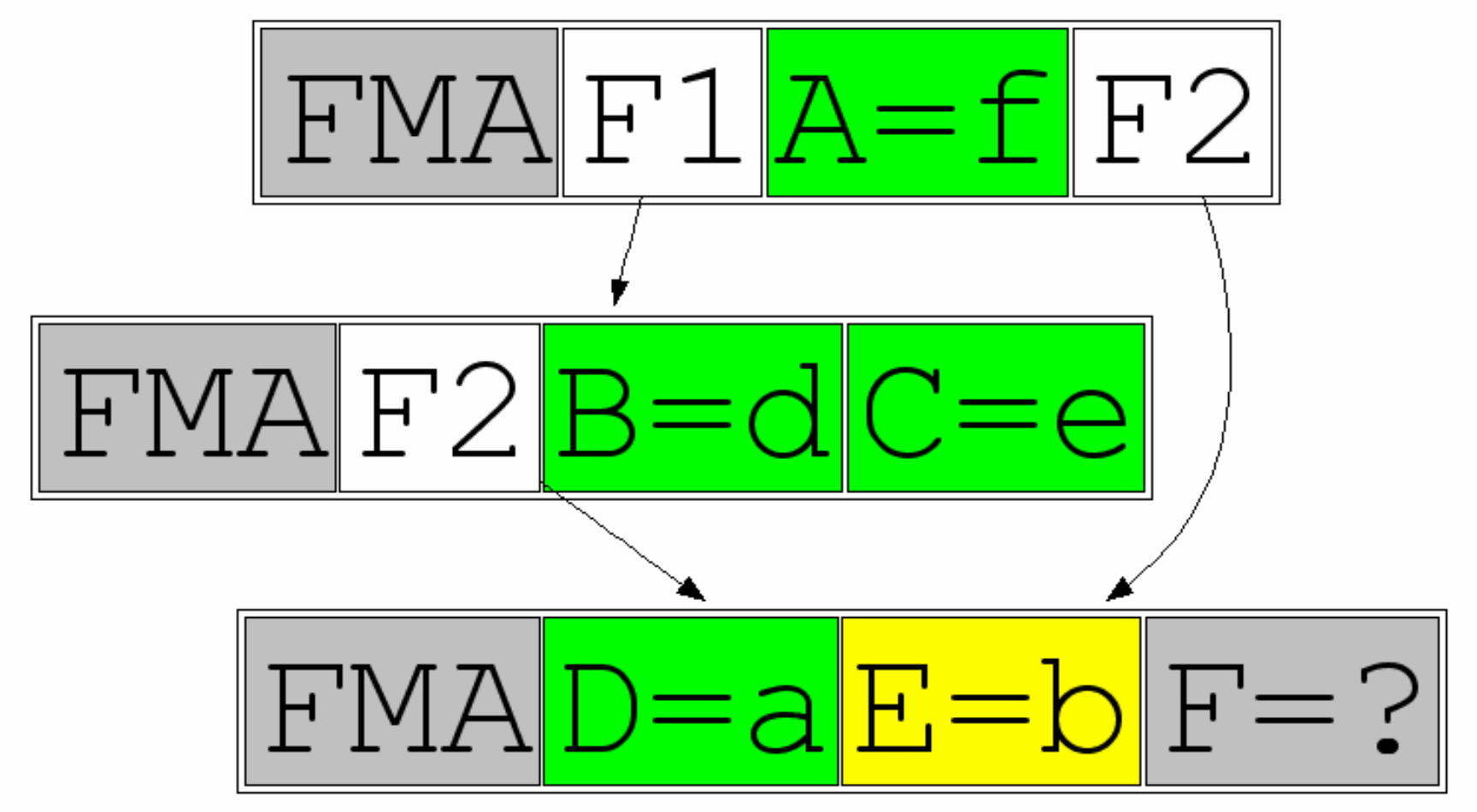
Formal: +ABDE+ABF+AC+DE+F
Actual: +abdf+cdf+ab-ef+c



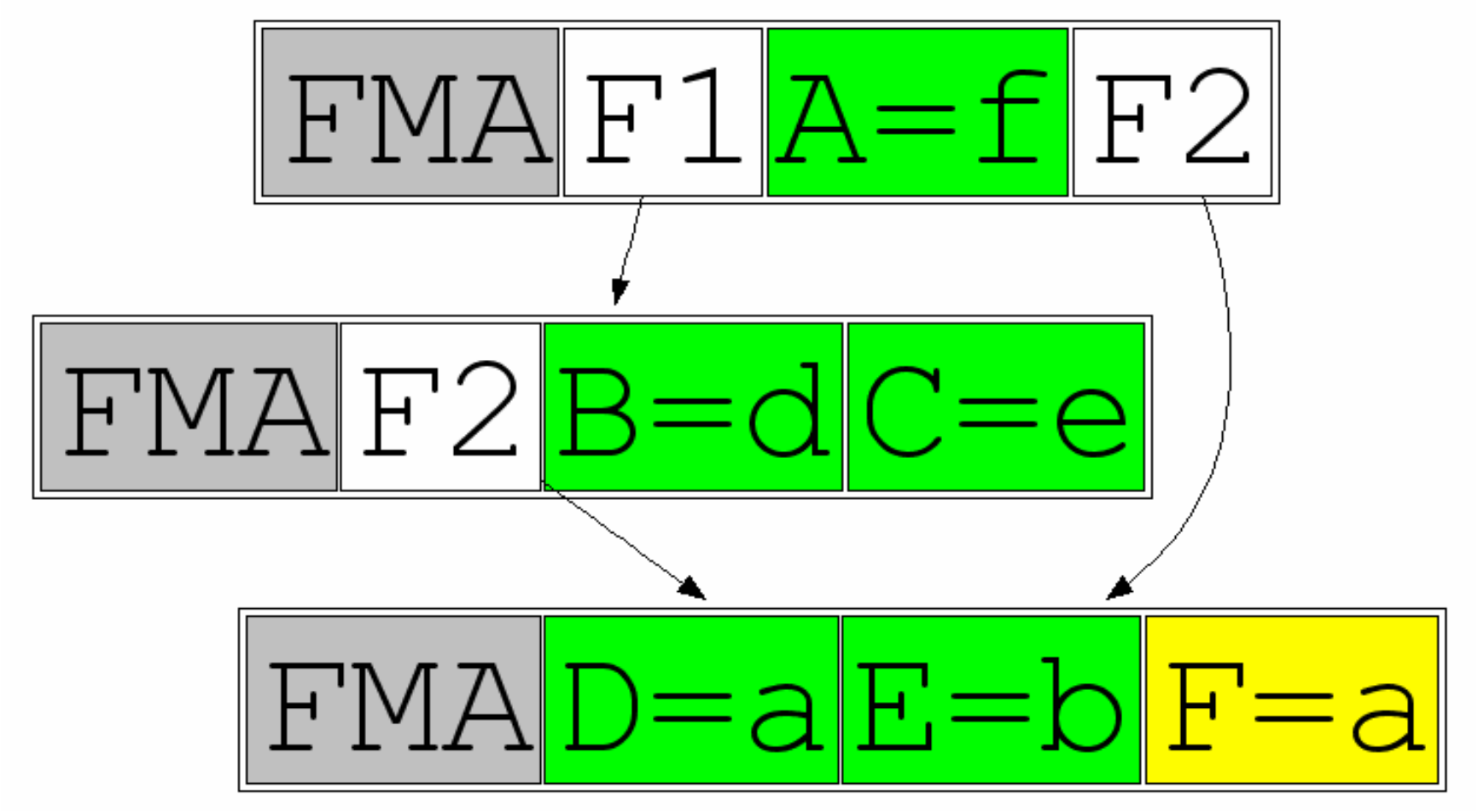
Formal: +ABDE+ABF+AC+DE+F
Actual: +abdf+cdf+ab-ef+c



Formal: +ABDE+ABF+AC+DE+F
Actual: +abdf+cdf+ab-ef+c

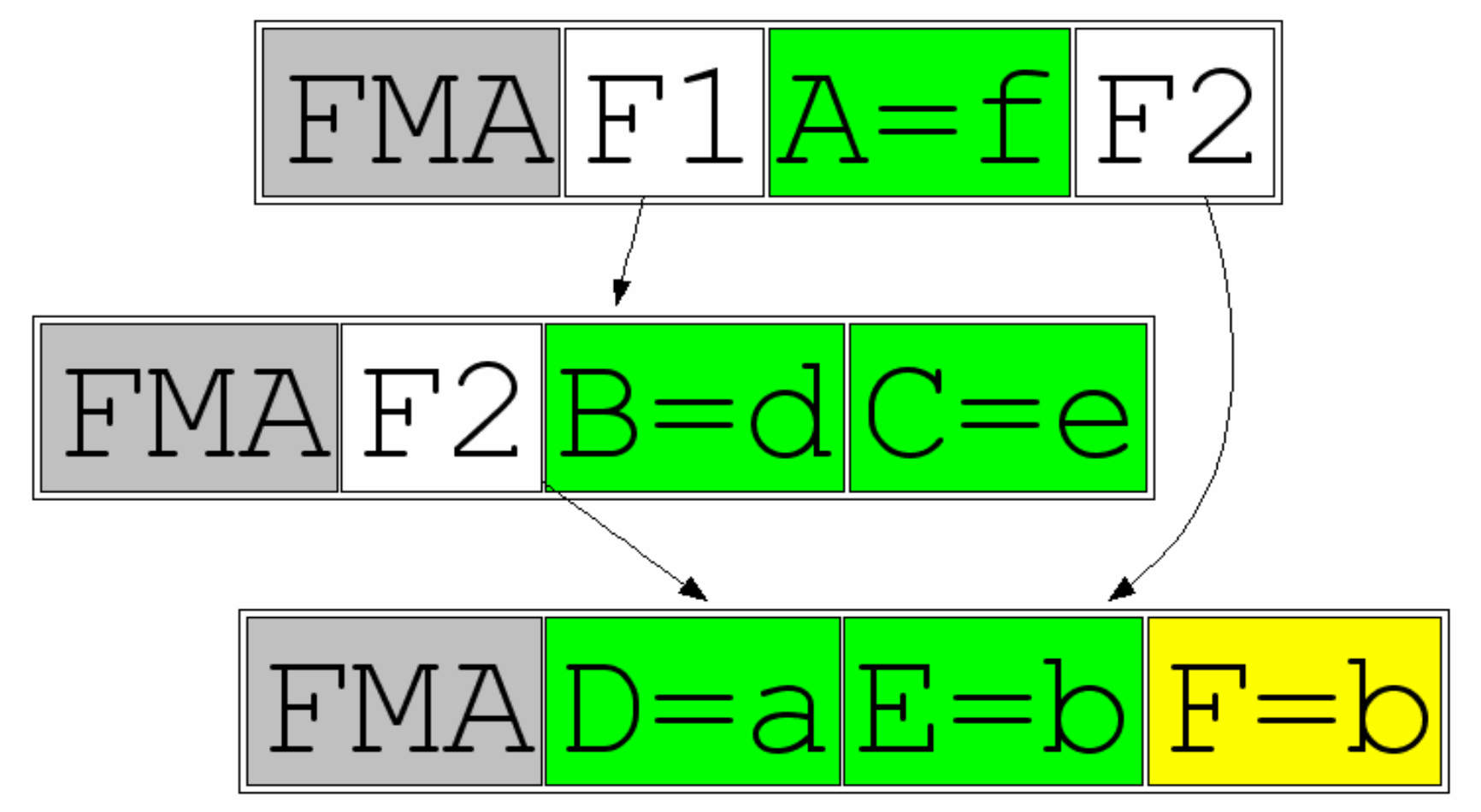


```
Formal: +ABDE+ABF+AC+DE+F
Actual: +abdf+cdf+ab-ef+c
```



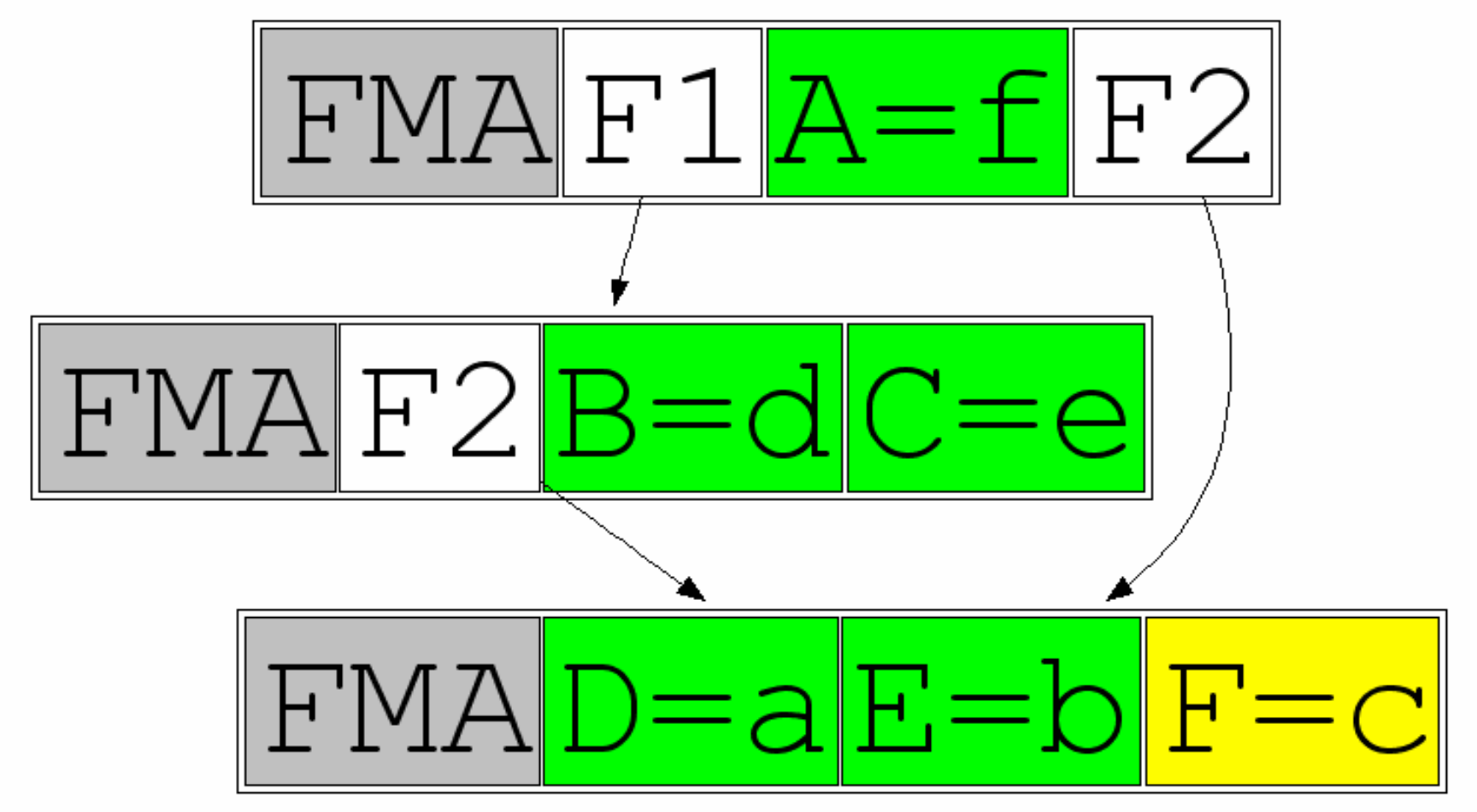
Formal: +ABDE+ABF+AC+DE+F

Actual: +abdf+cdf+ab-ef+c



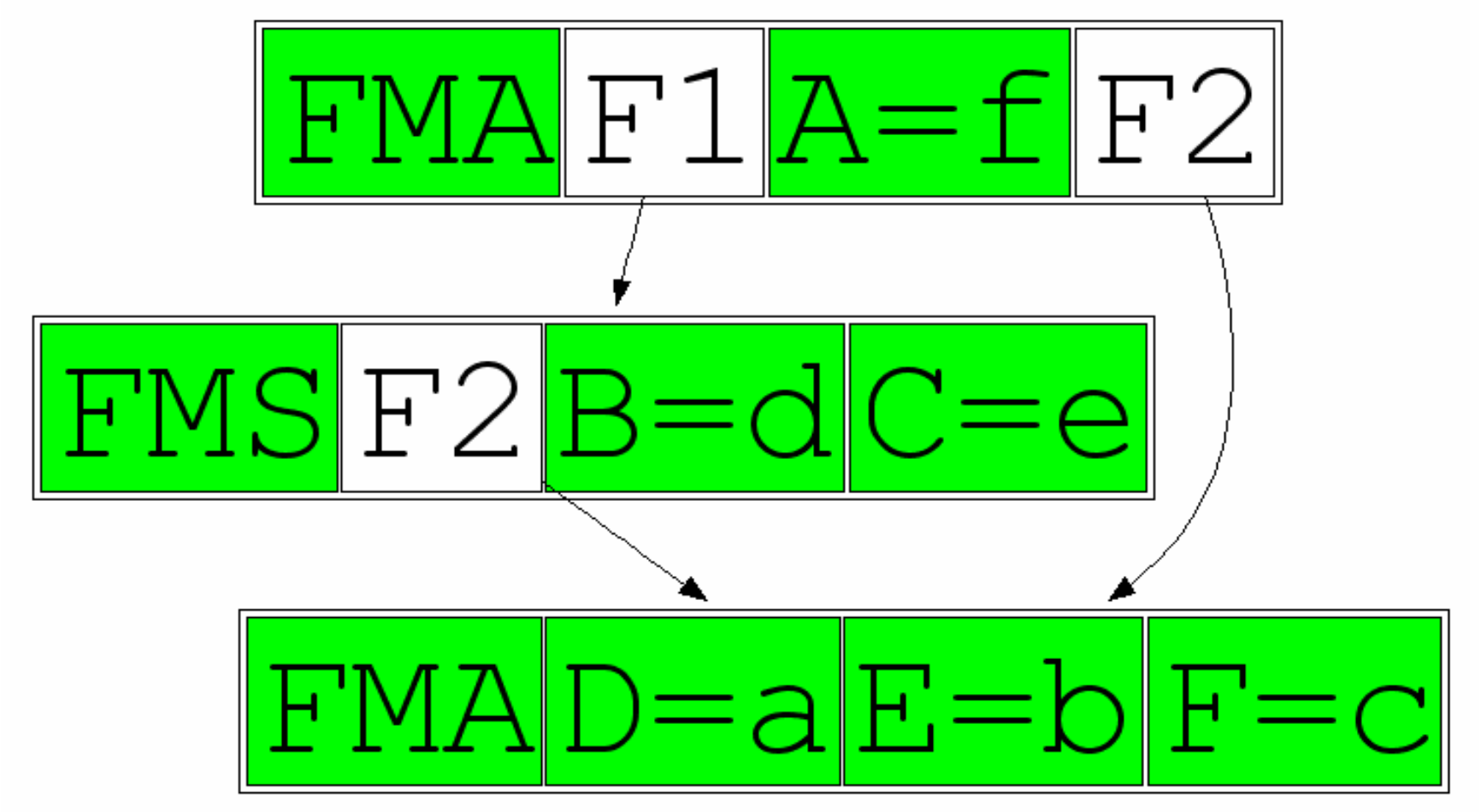
Formal: +ABDE+ABF+AC+DE+F

Actual: +abdf+cdf+ab-ef+c



Formal: +ABDE+ABF+AC+DE+F

Actual: +abdf+cdf+ab-ef+c



Results

- Performance gains:
  - 173.applu 11%
  - 435.gromacs 9%
  - 436.cactusADM 3%
  - 454.calculix 3%
- Optimization affects FP accuracy (too aggressive factorization and reassociation).
  - Enabled only under –IPF-fp-relaxed
- Compile time: no noticeable change.
  - Required to set a limit in breadth-first search to avoid rare exponential cases (**`-a-b-c-d-e-f`**)
- Patterns of bigger sizes do not give much except for few special cases (**a+b+c+d+e+f+g+h**)